# CrossTalk

Nov/Dec 2013 **The Journal of Defense Software Engineering** Vol. 26 No. 6

# Real-Time Information Assurance

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*

| | | |
|---|---|---|
| 1. REPORT DATE<br>**DEC 2013** | 2. REPORT TYPE | 3. DATES COVERED<br>**00-00-2013 to 00-00-2013** |

| | |
|---|---|
| 4. TITLE AND SUBTITLE<br>**CrossTalk, The Journal of Defense Software Engineering. Volume 26, Number 6. Nov/Dec 2013** | 5a. CONTRACT NUMBER |
| | 5b. GRANT NUMBER |
| | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | 5d. PROJECT NUMBER |
| | 5e. TASK NUMBER |
| | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)<br>**517 SMXS/MXDED,6022 Fir Ave,Hill AFB,UT,84056-5820** | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release; distribution unlimited**

13. SUPPLEMENTARY NOTES

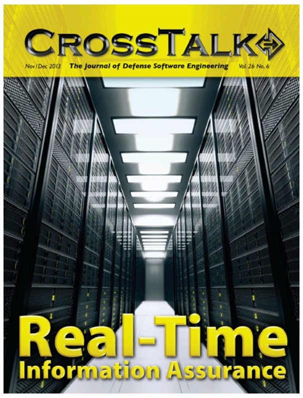
14. ABSTRACT

15. SUBJECT TERMS

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT<br>**unclassified** | b. ABSTRACT<br>**unclassified** | c. THIS PAGE<br>**unclassified** | **Same as Report (SAR)** | **44** | |

**Standard Form 298 (Rev. 8-98)**
Prescribed by ANSI Std Z39-18

TABLE OF CONTENTS


CrossTalk
Nov/Dec 2013 The Journal of Defense Software Engineering Vol. 26 No. 6
Real-Time
Information Assurance

Cover Design by
Kent Bingham

## Departments

## Real-Time Information Assurance

CrossTalk

| | |
|---|---|
| NAVAIR | Jeff Schwalb |
| DHS | Joe Jarzombek |
| 309 SMXG | Karl Rogers |

| | |
|---|---|
| Publisher | Justin T. Hill |
| Article Coordinator | Lynne Wade |
| Managing Director | David Erickson |
| Technical Program Lead | Thayne M. Hill |
| Managing Editor | Brandon Ellis |
| Associate Editor | Colin Kelly |
| Art Director | Kevin Kiernan |

| | |
|---|---|
| Phone | 801-777-9828 |
| E-mail | Crosstalk.Articles@hill.af.mil |
| CrossTalk Online | www.crosstalkonline.org |

**CrossTalk, The Journal of Defense Software Engineering** is co-sponsored by the U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Security (DHS). USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: Office of Cybersecurity and Communications in the National Protection and Programs Directorate.

**The USAF Software Technology Support Center (STSC)** is the publisher of CrossTalk providing both editorial oversight and technical review of the journal. CrossTalk's mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

**Subscriptions:** Visit <www.crosstalkonline.org/subscribe> to receive an e-mail notification when each new issue is published online or to subscribe to an RSS notification feed.

**Article Submissions:** We welcome articles of interest to the defense software community. Articles must be approved by the CrossTalk editorial board prior to publication. Please follow the Author Guidelines, available at <www.crosstalkonline.org/submission-guidelines>. CrossTalk does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the authors and their organizations.

**Reprints:** Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CrossTalk.

**Trademarks and Endorsements:** CrossTalk is an authorized publication for members of the DoD. Contents of CrossTalk are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

**CrossTalk Online Services:**
For questions or concerns about crosstalkonline.org web content or functionality contact the CrossTalk webmaster at 801-417-3000 or webmaster@luminpublishing.com.

**Back Issues Available:** Please phone or e-mail us to see if back issues are available free of charge.

CrossTalk is published six times a year by the U.S. Air Force STSC in concert with Lumin Publishing <luminpublishing.com>.
ISSN 2160-1577 (print); ISSN 2160-1593 (online)

## CROSSTALK would like to thank NAVAIR for sponsoring this issue.

As we see technology evolving, we are also bearing witness to the evolution of software development and software itself. These advances in software have been forged from the necessity of affordable improvements. From a programmatic perspective, it is not always cost effective to update hardware when there is an improvement. Therefore, we depend on software continuously leveraging that hardware to yield performance improvements. This ever-increasing reliance on software means that our requirements and the way we protect our systems cannot be a last-minute addition.

Newton's Third Law of Motion says that for every action there is an equal and opposite reaction. In the case of advancements in software, that opposite reaction is the advancement of threats against that software. Attacks continue to take advantage of technology and the dependence we have on software. In order to stay ahead of this curve, we must look to how we design and secure systems for tomorrow's threats.

The task of designing and securing systems is easier said than done. We are no longer dealing with network diagrams that can be drawn on one sheet of paper. Long gone are the days when your computer connected to a few other nodes in the network, then a single connection to the Internet. We are now faced with protecting systems that move in and out of multiple heterogeneous networks. The networks are scalable and rely more on near-real time or real time data. With multiple access points and ever changing users, we have to escalate to more than virus protection and firewalls.

When these nodes are aircraft and other military systems, the need for data on demand is even more critical. Information Assurance (IA) is therefore a key component to the protection and availability of our required data. However, we often regard the development of these IA requirements as an afterthought. Bringing the entire system performance in line with protecting its information has to be considered with both the risk and associated cost.

One approach to bring this field to the forefront is that of Real Time Information Assurance. This is increasingly important because our systems are in a constant state of flux. Changing users, conditions, incoming data and the need for that information to be processed and disseminated is making it harder to assign controls in the early stages of a project's design phase. We try to capture the flow of information in order to better secure it in the future. However, we cannot ever anticipate the demand and use of our systems at all times under all scenarios. Therefore, we need an adaptable manner in which to protect our systems depending on the complex state or states in which they are currently operating.

From the user perspective, the systems and the required information needed by those systems should be available at all times. With greater attention being focused on authentication, we cannot burden the users with additional wait times for validation. To the user, access of information and systems should be seamless. To an un-authorized user, access should be impossible or at the very least time consuming to the point that the information is no longer valid or useful. Analyzing these conditions in real time, can allow the best of both worlds.

The application of Real Time IA can aid a program in finding that ideal balance of cost, schedule and performance with security in mind. I hope you enjoy the articles in this issue of CrossTalk and consider their applicability to future programs. As you read, begin to think not only about Real Time IA, but to the future of Predictive IA.

**Felipe Jauregui**
**Chief Engineer, H-1 Weapons System Support Activity**
**NAVAIR**

# Integrating Information Assurance into Agile and Rapid Technology Development

# Agile IA

**Kathy McGinn, Marine Corps Systems Command**
**Monica Nelson, Marine Corps Systems Command**
**Dale Daigle, Space and Naval Warfare Systems Command**

**Abstract.** In this paper we present a method for integrating information assurance requirements into agile and rapid technology development. This integrated approach advocates involvement of the information assurance expert at the onset of system development and design—building requirements in proactively. The focus is on collaboration, continuous monitoring, and leveraging automated testing for formal validation.

## Introduction

As "agile" software development becomes more common in government and industry, it poses a challenge: integrating Information Assurance (IA) into system requirements and the development process. How do we take a gate-styled serial process, segment it into increments, and then integrate it? How do we move IA requirements from right to left in the development schedule, instead of tacking them onto the end? How do we document that process incrementally and obtain acceptance of the security posture by the approval authority? At Marine Corps Systems Command, the Information Assurance Management Team supporting Program Manager Marine Intelligence has developed an integrated information-assurance process, called "agile IA," to address these issues and better support agile software development.

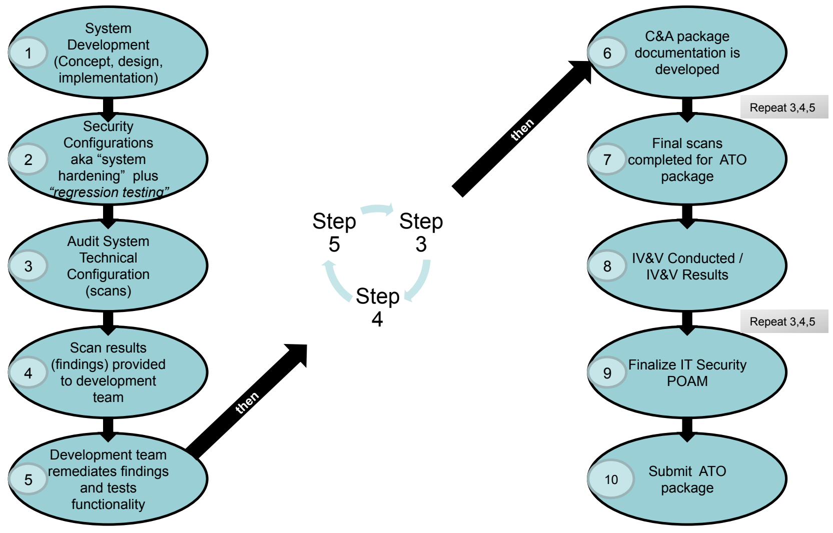

1 System Development (Concept, design, implementation)
2 Security Configurations aka "system hardening" plus "regression testing"
3 Audit System Technical Configuration (scans)
4 Scan results (findings) provided to development team
5 Development team remediates findings and tests functionality
then
Step 3
Step 4
Step 5
then
6 C&A package documentation is developed
Repeat 3,4,5
7 Final scans completed for ATO package
8 IV&V Conducted / IV&V Results
Repeat 3,4,5
9 Finalize IT Security POAM
10 Submit ATO package

*Figure 1: Legacy serial process of system development and preparation*

Typically, legacy certification and accreditation practices assess IA requirements after a system has been designed and developed, generally through serial processes that prolong the development schedule. This results in reactionary implementation of security configurations and creates a cascading effect of schedule slips and unplanned costs. This traditional approach can leave both technical and administrative security requirements inadequately addressed. It frequently overlooks operations security—even core requirements such as ensuring that personnel are properly cleared.

Legacy processes involve cycles of configuration known as system "hardening." The cycle consists of scanning, remediating, rescanning and reconfiguration, in multiple iterations whose goal is to minimize vulnerability. The baseline must be maintained while the system is documented and formal, Independent Verification and Validation (IV&V) is conducted. IV&V likewise involves cycles of activity that must occur while other processes are accomplished for Certification and Accreditation (C&A). The combined process of hardening and IV&V can take anywhere from 12 to 24 months—occasionally even longer.

Figure 1 depicts the average legacy serial process of preparing a system for obtaining authority to operate. Typically, the IA subject matter expert, such as the information-system security manager or information system security engineer, is called into the effort at step 3.

The goal of system development is to deliver a capability to an end user—in the case of the U.S. Marine Corps, to the warfighter. But legacy practices delay delivery, and for some systems—such as intelligence sensor systems—delay is itself a security risk. As stated in a recent paper by the SEI at Carnegie Mellon University, "the information assurance accreditation delay is so extensive (often months to a year) that [the gated implementation method of] the DIACAP process almost negates the benefits gained through rapid development methods."[1]

## Collaborate, Integrate, Automate

The integrated or "agile IA" approach encourages a rapid, flexible response to vulnerabilities that emerge as the system is developed. The key difference between agile IA and the legacy approach is the integration of security at the onset of system development, allowing proactive attention to requirements. The IA subject-matter expert is involved in development from day one, and is part of the agile development team, which addresses and prioritizes IA requirements during sprint planning. Automated scanning, which includes automated code-review tools, allows the expert to monitor the system continuously as it is being developed. As security requirements are identified, they are reported in the information-technology security plan of actions and milestones. By "baking" security into the product, agile IA reduces security risk at any point in development and minimizes redundancy in the hardening process. Figure 2 reflects the cycle of continuous monitoring and proactive remediation and tracking.

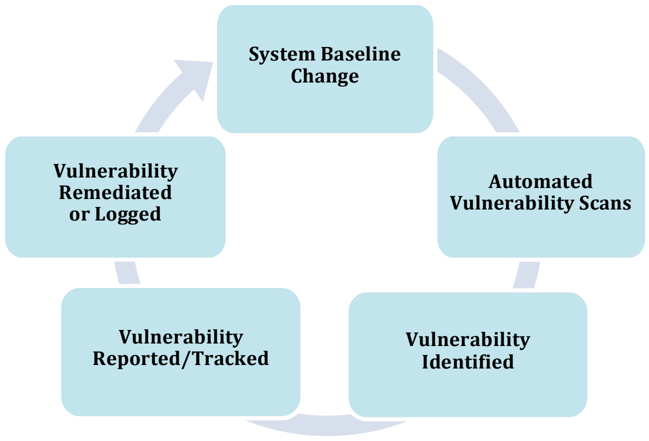

System Baseline Change
Automated Vulnerability Scans
Vulnerability Identified
Vulnerability Reported/Tracked
Vulnerability Remediated or Logged

*Figure 2 – Continuous monitoring and remediation tracking process*

C&A also poses the usual timeline challenges. In the agile approach, artifacts are created incrementally as the system develops. IA controls are documented as the information becomes available. Documentation is limited to the minimum required to satisfy the approving authority requirement to depict the system security posture.

Lastly, and more significantly, a certified validator is engaged to leverage the results of the continuous monitoring already occurring during each sprint. This enables the validator to leverage the test results for formal validation and verification, and perform a shortened “hybrid” style of IV&V. The formal IV&V event is reduced to minimally disruptive administrative checks and audits, obviating the intensive auditing that typically interrupts system development. System development is allowed to continue during the IV&V audit.

Figure 3 depicts how continuous monitoring can be tracked to reflect the changing baseline of a system given its vulnerability findings. Note that in the beginning of the monitoring, the number of findings is high. As the system is developed and remediation occurs, findings are reduced. However, fluctuations in the baseline should be expected as system requirements are added.

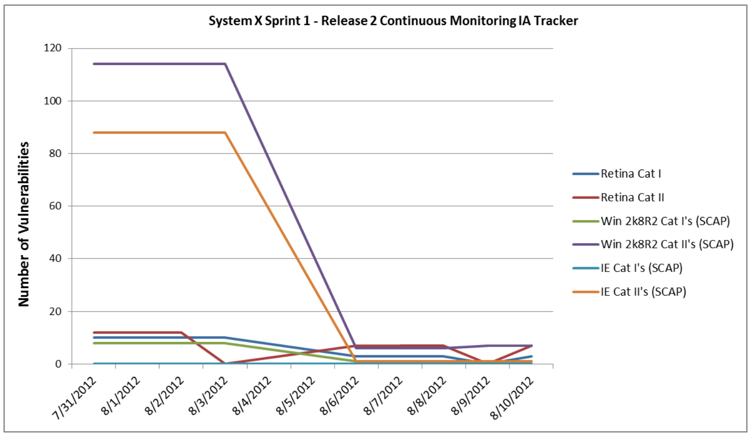


*Figure 3 – Continuous monitoring scan results example*

## Conclusion

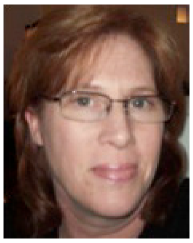
Integration of security configuration requirements at initial stages of system development eliminates the legacy practice of reactionary IA implementation. Introducing IA expertise at the onset of development provides a more holistic approach to identifying and implementing security requirements. The agile IA approach can be implemented within other development methods and should not be practiced for agile development alone. It allows an incremental and controlled approach for IA implementation and ensures a more secure posture of a capability or product release. Figure 4 depicts a sprint timeline with concurrent activities per sprint to include security configuration, functional testing, continuous monitoring, formal auditing, and incremental C&A package creation.

Integrated IA will reduce the schedule slippage and unplanned costs that are inherent when IA is bolted on at the end of system development. It will also reduce redundancy in system-hardening processes, leverage auditing activity for multiple purposes, and produce an artifact package that more quickly depicts the system's security posture. More importantly, it will ensure that capabilities or products provided to the warfighter are less vulnerable to exploitation.✧

## NOTES

1. Bellomo, Stephany; Woody, Carol, Software Engineering Institute at Carnegie Mellon, Paper: DOD Information Assurance and Agile: Challenges and Recommen dations Gathered through Interviews with Agile Program Managers and DOD Accreditation Reviewers, Technical note CMS/SEI-2012-TN-024

## ABOUT THE AUTHORS

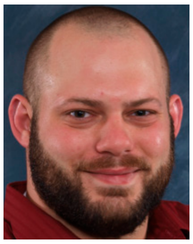


Kathy McGinn is the lead Information System Security Manager for the Marine Intelligence program office at the Marine Corps Systems Command in Quantico, Virginia. She currently serves as an IA subject matter expert; managing certification and accreditation requirements for multiple programs. She has extensive knowledge and experience in information assurance process development and certification and accreditation. She is a certified information systems security professional (CISSP).

**E-mail: kathy.mcginn@usmc.mil**

Monica S. Nelson is an Information System Security Manager for the Marine Intelligence program office at the Marine Corps Systems Command in Quantico, Virginia. She currently serves as the "agile IA" subject matter expert for her team. She has extensive knowledge and experience with agile software development and is a member of the team that developed the first integrated information assurance process using Agile methodologies. She holds the CompTIA Security+ certification and has a Bachelor's degree in Decision Science and Management Information Systems from George Mason University.

**E-mail: monica.nelson@usmc.mil**



Dale Daigle is the lead Information System Security Engineer (ISSE) for Distributed Common Ground/Surface System - Marine Corps (DCGS-MC) at Space and Naval Warfare Systems Command (SPAWAR) in Charleston, South Carolina. He currently serves as a subject matter expert on Information Assurance (IA) and "Agile IA" for DCGS-MC. He has extensive knowledge and experience with IA and agile software development and led efforts to develop the first integrated IA process using Agile methodologies. He is a Certified Information Systems Security Professional (CISSP).

**E-mail: dale.daigle@navy.mil**

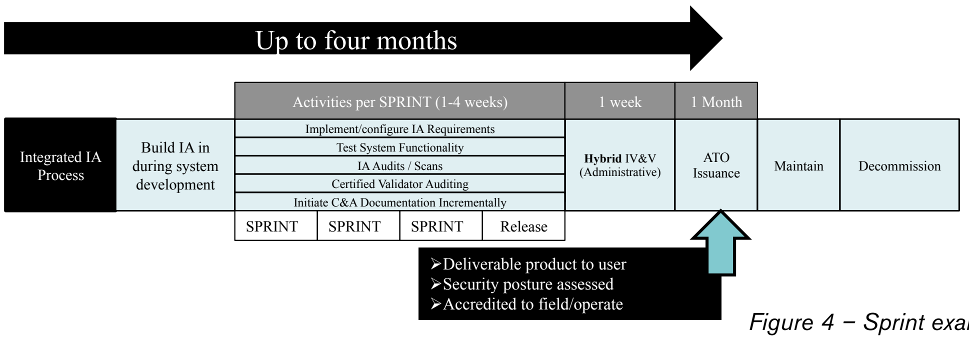

Up to four months
Activities per SPRINT (1-4 weeks)
1 week
1 Month
Integrated IA Process
Build IA in during system development
Implement/configure IA Requirements
Test System Functionality
IA Audits / Scans
Certified Validator Auditing
Initiate C&A Documentation Incrementally
SPRINT
SPRINT
SPRINT
Release
Hybrid IV&V (Administrative)
ATO Issuance
Maintain
Decommission
➢Deliverable product to user
➢Security posture assessed
➢Accredited to field/operate

*Figure 4 – Sprint example with concurrent integrated activity*

# Disciplined Agile Delivery

## The Foundation for Scaling Agile

**Scott W. Ambler, Scott Ambler + Associates**
**Mark Lines, Scott Ambler + Associates**

**Abstract.** Many organizations have adopted agile strategies to some extent, often applying simple methods such as Scrum on a few software development projects. Although they have succeeded on a handful of projects—clearly a good start—they now want to scale agile to address enterprise-class solutions. As the old saying goes about success, "What got you here is not going to get you to the next level."

This article explores several important issues when it comes to applying agile strategies at scale. First, it explores what it means to scale agile. Our experience is that without this understanding it is incredibly difficult to come to an agreement as to how to tailor agile strategies to meet your needs at scale. Second, we overview the Disciplined Agile Delivery (DAD) process decision framework and show how it provides a solid foundation from which to scale agile. Third, we explore how DAD's (process) goal-driven approach is the key for scaling agile solution delivery. Fourth, we warn you of the dangers of assuming that you just need to adopt a few new practices. Fifth, we review some survey-based evidence that summarizes the experiences of the industry when it comes to applying agile at scale.

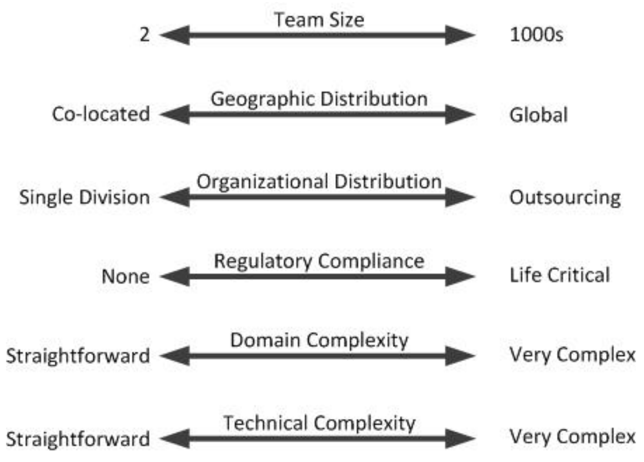


*Figure 1. An overview of process scaling factors.*

## Context Counts

There is more to scaling than the size of the team. Figure 1 depicts what we have found to be the six key factors when scaling agile—team size, geographic distribution, organizational distribution, regulatory compliance, domain complexity, and technical complexity—all of which are ranges. These six factors, a simplification of the Agile Scaling Model (ASM) [1] that Scott W. Ambler led the development of within IBM, form the scaling portion of what we call the Software Development Context Framework (SDCF). The process selection portion of the SDCF is comprised of four factors (team skills, team culture, organizational culture, and problem type) that are also ranges. The process selection factors are similar to the environment risk factors first described by Boehm and Turner [2] and the combination of the process selection and scaling factors are similar to Kruchten's situational agility octopus [3]. The fundamental point is that context counts – one strategy does not fit all.

Taken together, the process selection factors and the scaling factors drive the way that a team will tailor its organization structure, its process, and its work environment. Each team will find itself in a unique situation and will need to tailor their strategy accordingly, a potential challenge if your organization still clings to a "repeatable process" philosophy. For example a team of seven co-located people in a regulatory environment will work differently than a team of 40 people spread out across several locations in a non-regulatory environment.

There are three challenges with a context-driven approach. First, few organizations want their solution delivery teams creating their own processes on the fly as this is very inefficient both at the team level and the organization level. Clearly there is need for a common starting point. Second, contrary to what a team may believe, they are unlikely to have the agile expertise required to tailor a strategy that works well for them. In this case there is need for some context-driven guidance. Third, many organizations, particularly in the public and IT service provider sectors, must work in a manner that is CMMI® compliant. When the team and organizational cultures are sufficiently flexible this is definitely possible although it does require some out-of-the-box thinking for everyone involved [4]. To address these challenges, and more, you need to adopt a disciplined approach to agile solution delivery.

## Disciplined Agile Delivery (DAD)

From the summer of 2006 to the summer of 2012 Scott W. Ambler worked in the role of Chief Methodologist for IT at IBM Rational, working with organizations around the world to apply agile and lean techniques at scale. The DAD process decision framework was developed as the result of the observation that there were common patterns for applying agility at scale successfully. The DAD framework reflects the experiences of IBMers working in the field with customer organizations, IBMers applying agile at scale internally, and business partners such as Mark Lines who were also working with customer organizations.

DAD is a second-generation framework that strives to provide a coherent, end-to-end strategy for how agile solution delivery works in practice. DAD is a people-first, learning-oriented hybrid agile approach to IT solution delivery. It has a risk-value lifecycle, is goal-driven, is scalable, and is enterprise aware. Although all of these characteristics are important, several of them are critical for scaling agile:

**1. Hybrid.** DAD is a hybrid in that it adopts and tailors proven strategies from methods such as Scrum, Extreme Programming (XP), Agile Modeling (AM), Unified Process (UP), Kanban, Outside In Development (OID), and Agile Data (AD) to name a few. Instead of starting with a process kernel such as

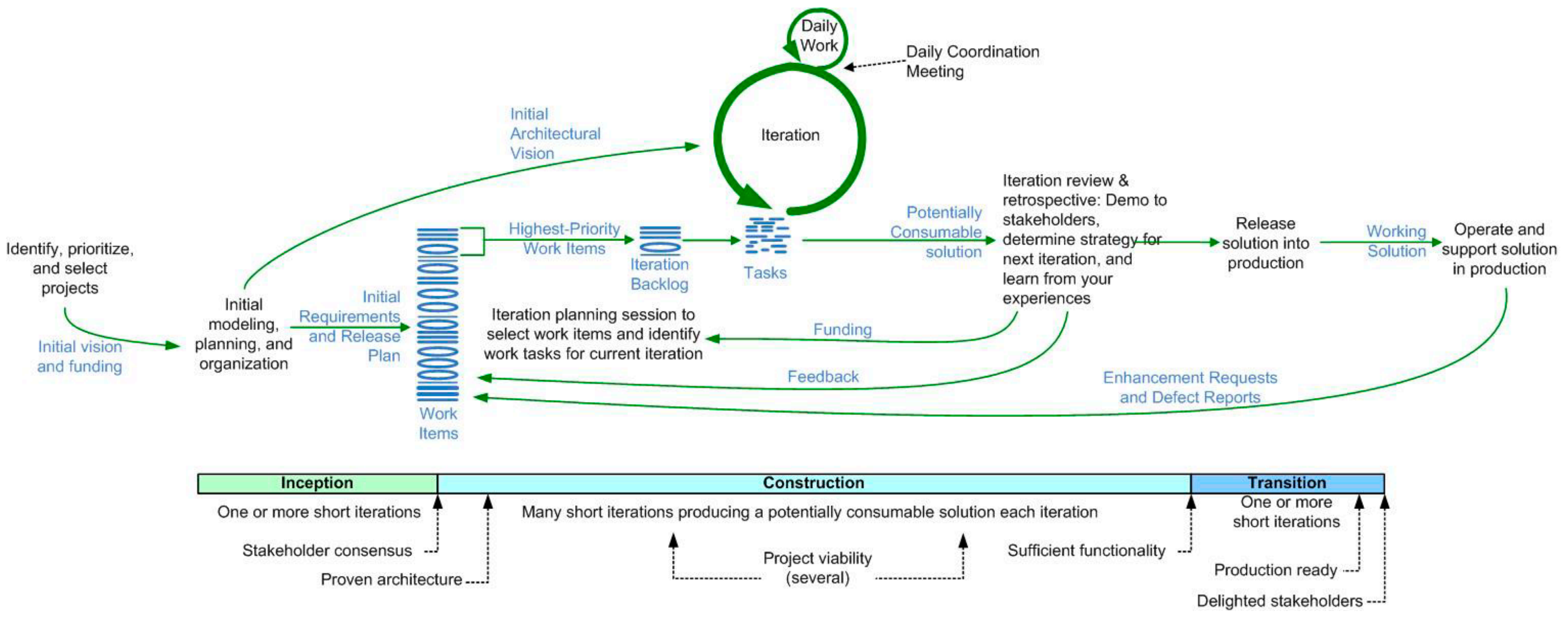

Daily Work
Daily Coordination Meeting
Initial Architectural Vision
Iteration
Identify, prioritize, and select projects
Initial vision and funding
Initial modeling, planning, and organization
Initial Requirements and Release Plan
Work Items
Highest-Priority Work Items
Iteration Backlog
Tasks
Potentially Consumable solution
Iteration review & retrospective: Demo to stakeholders, determine strategy for next iteration, and learn from your experiences
Release solution into production
Working Solution
Operate and support solution in production
Iteration planning session to select work items and identify work tasks for current iteration
Funding
Feedback
Enhancement Requests and Defect Reports
Inception
Construction
Transition
One or more short iterations
Many short iterations producing a potentially consumable solution each iteration
One or more short iterations
Stakeholder consensus
Proven architecture
Project viability (several)
Sufficient functionality
Production ready
Delighted stakeholders

*Figure 2: The basic lifecycle for DAD.*

Scrum and then adding in a large number of other practices and techniques, or starting with a comprehensive process framework such as the UP and tailor it down to something usable, why not simply start in the middle which is where you actually need to get to anyway? The upshot is that with DAD much of the expensive and time-consuming work of combining agile techniques has already been done for you.

**2. Enterprise aware.** When you are working at scale you cannot afford to have a delivery team take a stovepipe mentality and work on their own, even when they are producing "potentially shippable software" on a regular basis. The reality is that agile delivery teams do not work in a vacuum. There are often existing systems currently in production, and minimally your solution should not impact them although hopefully your solution will leverage existing functionality and data available in production. There are often other teams working in parallel to your team, and you may wish to take advantage of a portion of what they are doing and vice versa. There may be a common vision that your organization is working towards, or that your program is working towards, a vision that your team should contribute to. There will be a governance strategy in place, hopefully an agile/lean one that enhances what your team is doing—in fact, when we developed DAD we invested significant effort describing how to effectively measure and govern agile teams [5]. Disciplined agile teams recognize that they are part of a larger, organizational ecosystem and act accordingly.

**3. Solution focused.** With DAD you mature your focus from just producing software to instead providing consumable solutions that provide real business value to your stakeholders within the appropriate economic, cultural, and technical constraints. Yes, software is clearly important, but in addressing the needs of our stakeholders we will often provide new or upgraded hardware, change the business/operational processes that stakeholders follow, and even help change the organizational structure in which our stakeholders work. When you are working at scale you cannot afford to fall into the "potentially shippable software" trap.

**4. Delivery focused.** The basic DAD lifecycle, depicted in Figure 2, addresses the project lifecycle from the point of initiating a project through construction to the point of releasing the solution into production (it also shows some pre-initiation portfolio management activities as well as post-delivery production activities). This differs from first generation agile methods that typically focus on the construction aspects of the lifecycle, leaving the details about how to perform the rest of it up to you. In fact, you can see how the construction portion of the basic DAD lifecycle reflects an improved version of the Scrum lifecycle. Lightweight milestone reviews, one aspect of DAD's support for agile governance, are depicted along the bottom of the lifecycle. More importantly, because DAD is not prescriptive, the lifecycle of Figure 2 is only one of several supported by DAD. There are also lean/advanced and continuous delivery versions of the lifecycle that abandon many of the constraints prescribed by Scrum.

**5. Goal-driven.** Every team finds itself in a unique situation, often facing the risks associated with one or more of the scaling factors discussed earlier. To address this challenge the DAD process framework takes a process goal-driven approach where a team tailors their strategy to reflect the context of the situation they find themselves in. Instead of saying that you should organize your work as a stack prioritized by business value (what Scrum prescribes) DAD instead says that you need to manage changing stakeholder needs in some way. DAD also describes how you have several ways of addressing this goal (the Scrum product backlog being one of them) and that there are advantages and disadvantages to each. A goals-driven approach provides disciplined agile teams with the guidance they require to tailor their approach to appropriately address the scaling factors that they face. More on this later.

## Goal Driven

Very likely the most intriguing aspect of the DAD process decision framework is the fact that it is goal driven instead of prescriptive. Figure 3 summarizes the DAD process goals from a lifecycle point of view. These goals are applicable regardless of the situation faced by a team, but the way that the goals are addressed may vary substantially.

Let us do a deep dive into a single goal. Figure 4 presents a goal diagram for identifying an initial technical strategy, something that you do fairly early in a project. The goal is depicted as a rounded rectangle, issues that you should consider as normal rectangles, and potential strategies to address an issue as a list. Some strategy lists are prioritized in order of agile preference, something that is indicated with an arrow to the left of the list (as you see in Figure 4 with Level of Detail strategies). The DAD framework also recommends starting points; these strategies are shown in bold and italics, so as to provide guidance to teams new to agile.

DAD's goal-driven approach underlies the idea that to be effective at applying agile a team must understand the context in which they are working. For example, a team that is co-located has the option of capturing their technical strategy using a few whiteboard sketches via informal modeling sessions. However, a team that is globally distributed would likely need to take a detailed interface approach, a strategy called API First in the Eclipse Way and Contract Modeling in Agile Modeling, to overcome the communication challenges surrounding geographic distribution. A team working in a rigid organizational culture or in a regulatory environment may decide on formal modeling sessions over informal ones. A team facing significant technical complexity may want to consider several candidate architectures to increase their chance of success.

The point is that different teams face different situations; therefore they will need to adopt their strategy to reflect the situation. Each team needs to identify an initial technical strategy, explore their initial scope, develop an initial plan, and fulfill many other goals but they will achieve these goals in different ways. The DAD process framework provides straightforward guidance to help you to make these tailoring decisions effectively. It does this by explicitly describing the process decision that you are making and then walks you through the process of making it. It does this in a two-fold manner, first by overviewing common options available to you in a visual manner via goal diagrams and second by describing the advantages and disadvantages of each option via a table-based approach.

## More than Just a Few Extra Practices

Yes, you are going to add some practices over and above what a small, co-located agile team typically does. For example, although architecture is always important, it is critical at scale, thus DAD adopts architecture-oriented practices from several sources. From AM there are practices as initial architecture envisioning and just-in-time model storming throughout the project, from the UP there is proving the architecture early in the lifecycle with working code, and from XP architecture spikes, to name a few. These agile architecture practices not only help you to get started on the right foot they help you to avoid taking on unnecessary technical debt which otherwise would bog your team down.

You will also find that you need to adopt a more sophisticated approach to testing that goes beyond the whole-team strategy favored by agile-in-the-small teams. This includes parallel independent testing, reviews (both formal and informal), and even end-of-lifecycle testing in some cases. Similarly your requirements strategy will become more sophisticated. It should come as no surprise that user stories written on index cards do not scale well.

And you will need to go beyond some of the simplistic team organization advice. Scrum's three roles work well in some situations but you will find DAD's approach of five primary roles supported by another five secondary roles will support scaling more effectively. DAD introduces primary roles of Architecture Owner (an agile solution architect) and Stakeholder to supplement the Scrum-like roles of Team Lead (ScrumMaster), Product Owner, and Team Member. The secondary roles of Domain Expert, Independent Tester, Integrator, Technical Expert, and Specialist address the complex environment faced by teams working at scale. Furthermore, this more robust set of roles makes it easier to transition to agile because it is clearer how existing experienced staff can still add value on their team. Coordination on large teams requires more than a 15-minute "scrum of scrums." Instead, you will find that your team leads will coordinate project management issues, your product owners requirements issues, and your architecture owners technical issues. This leadership team will typically be headed up by a program management specialist [5].

| Goals for the Inception Phase | Goals for Construction Phase Iterations | Goals for the Transition Phase |
|---|---|---|
| - Form initial team | - Produce a potentially consumable solution | - Ensure the solution is consumable |
| - Develop common project vision | - Address changing stakeholder needs | - Deploy the solution |
| - Align with enterprise direction | - Move closer to deployable release | |
| - Explore initial scope | - Improve quality | |
| - Identify initial technical strategy | - Prove architecture early | |
| - Develop initial release plan | | |
| - Form work environment | | |
| - Secure funding | | |
| - Identify risks | | |
| **Ongoing Goals** | | |
| - Fulfill the project mission | - Improve team process and environment | |
| - Grow team members | - Leverage and enhance existing infrastructure | |
| - Address risk | - Coordinate activities | |

*Figure 3. Goals addressed throughout a DAD project.*

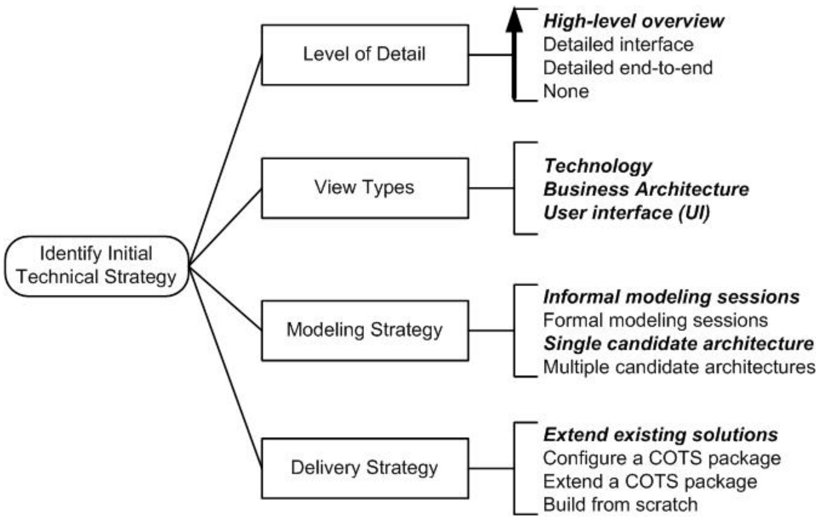


*Figure 4. Goal diagram: Identify Initial Technical Strategy*

The scaling factors we described earlier should make it clear that there is more to scaling agile than adopting a few additional practices. You saw in the discussion around how to tailor your approach to identifying an initial technical strategy that various scaling factors will change how you approach certain issues. The level of detail in your technical strategy will vary, the way in which you approach modeling will vary, the views you capture may vary (each of the architecture view types listed in Figure 4 break down into multiple views), and even how you choose to deliver the solution may vary. And this is one of the simpler goals to understand, which is why we used it as an example. Construction goals of Produce a Potentially Consumable Solution and Move Closer to Deployable Release are much more complex and thus trickier to tailor.

## Are Organizations Succeeding With Agility at Scale?

Heck yes! In the summer of 2012, Scott W. Ambler ran a survey for Dr. Dobb's Journal—a follow up to a similar 2009 survey, which explored whether organizations were successful at applying agility at scale [6]. The survey was designed to determine whether organizations were attempting to apply agile techniques at scale, whether anyone was succeeding at doing so, and whether anyone was struggling to do so. The answers to those three questions, respectively, are yes, yes, and yes. Although most agile teams are 20 people or less, some organizations are applying agile techniques on team sizes of several hundred. Organizations are successfully applying agile techniques at all levels of geographic distribution, at all levels of organizational distribution, and at all levels of domain and technical complexity. They are even succeeding at applying agile in regulatory environments, including life critical ones.

Where some organizations are succeeding with agility at scale, the survey also found that some organizations are unfortunately struggling to do so, the implication being that it is not a slam dunk after all. Our experience has been that the teams that get into the most trouble when applying agility at scale are the ones that are still trying to apply strategies geared for small, co-located teams in relatively straightforward situations. Another common failure pattern seems to be overly focused on construction-phase issues while underestimating the challenges associated with initiating an agile project successfully. A third common failure pattern is not involving at least a few people on the team with experience applying agile successfully in similar situations.

Woodward et. al. [7] describes the experiences of several IBM teams at applying agile at scale, particularly in geographically distributed situations as well as on large teams often in complex scenarios. Several teams were globally distributed, several teams numbered in the hundreds of developers, and several teams had both of these attributes. In some cases the teams were working on existing products with millions of lines of legacy code that had been written years, and sometimes decades earlier, using non-agile techniques. In all cases the IBM teams found they needed to tailor a hybrid, disciplined agile approach that reflected the situation they found themselves in. Alan Brown [8] describes his experiences helping organizations to scale agile, including a European financial institution applying early versions of the DAD framework. This organization successfully adopted and tailored agile delivery strategies at scale with project teams that were "in flight" as they could not simply shut down IT for several months while they retooled their staff to become agile.

Other surveys have explored how scaling factors affect project success rates [9]. For example as team size gets larger, the success rate goes down—something which is true for all development paradigms. Additionally, as a team becomes more geographically distributed, the success rate corresponding drops—something that is also true for all development paradigms. Perhaps more importantly the surveys have found that regardless of size or geographic distribution, the success rate of agile teams is statistically as great or greater than that of traditional teams. If your organization is still struggling with the decision as to whether they should apply agile at scale they may take comfort in this observation.

An interesting aspect of these surveys is that they did not force a single definition of success, such as "on time and on budget", on the respondents. Instead they asked respondents to answer the questions from the point of view of the actual success criteria for the projects. The surveys were designed this way to reflect the fact that there is no one single definition of project success but instead it varies between teams. For all of these surveys the questions as they were originally asked and the source data from the survey are available free of charge. If you do not trust our analysis of the data you are welcome to do your own.

## You Need Greater Discipline

In this article we argued that there are many factors, not just team size, to consider when scaling agile. We also argued that you need an adequate foundation from which to scale agile, and that foundation we believe is DAD. The DAD process decision framework defines a full delivery lifecycle, showing how to deliver an agile project from start to finish, which is a hybrid of existing agile methodologies and techniques. DAD promotes an enterprise-aware, governed, goal-driven approach to agile solution delivery. DAD's goal-driven nature is the key to tailoring it to support agility at scale. Although many organizations are successfully applying agile strategies at scale, at the same time many organizations are getting into trouble doing so. The teams that run aground at scale invariably do so because they apply the simplistic approaches espoused by other agile methods in situations where they have little hope of success. It is definitely possible, and desirable, to apply agile techniques at scale—you just need to take a disciplined approach to doing so.

## Acknowledgments:

Material for this article was summarized from the book Disciplined Agile Delivery: A Practitioners Guide to Agile Software Delivery in the Enterprise (IBM Press, June 2012) by Scott W. Ambler and Mark Lines. For more about the DAD framework, visit <http://www.DisciplinedAgileDelivery.com>.

## ABOUT THE AUTHORS

Scott W. Ambler is a Senior Consulting Partner at Scott Ambler + Associates and he helps organizations around the world to help them to improve their software processes. He is the founder of the Agile Modeling (AM), Agile Data (AD), Disciplined Agile Delivery (DAD), and Enterprise Unified Process (EUP) methodologies and creator of the Agile Scaling Model (ASM). Scott is the (co-)author of 21 books, including Disciplined Agile Delivery, Refactoring Databases, Agile Modeling, Agile Database Techniques, The Object Primer 3rd Edition, and The Enterprise Unified Process.

**E-mail: Scott@scottambler.com.**

Mark Lines is the Senior Managing Partner at Scott Ambler + Associates. He is a disciplined agile black belt and mentors organizations on all aspects of software development. He is passionate about reducing the huge waste in most IT organizations, and demonstrates hands-on approaches to speeding execution and improving quality with agile and lean techniques. Mark provides IT assessments, and executes course corrections to turn around troubled projects. He writes for many publications and is a frequent speaker at industry conferences.

**E-mail: Mark@scottambler.com.**

## REFERENCES

1. Ambler, S.W. (2009). The Agile Scaling Model: Adapting Agile Methods for Complex Environments. IBM Whitepaper. <ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/raw14204usen/RAW14204USEN.PDF Retrieved Jan 2, 2013>.
2. Boehm, B.W. and Turner, R. (2003). Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley/Pearson Education.
3. Kruchten, P (2012). The frog and the octopus: a conceptual model of software development. <http://arxiv.org/abs/1209.1327> Retrieved Jan 2, 2013.
4. McMahon, Paul E. (2010). Integrating CMMI and Agile Development: Case Studies and Proven Techniques for Faster Performance Improvement. Addison-Wesley Professional.
5. Ambler, S.W. and Lines, M.J. (2012). Disciplined Agile Delivery: A Practitioner's Guide to Agile Solution Delivery in the Enterprise. IBM Press.
6. Ambler, S.W. (2012). Agility at Scale Survey: Results from the Summer 2012 Dr. Dobb's Journal State of the IT Union Survey. <http://www.ambysoft.com/surveys/stateOfITUnion201209.html> Retrieved Jan 2, 2013.
7. Woodward, E., Surdek, S., and Ganis, M. (2010). A Practical Guide to Distributed Scrum. IBM Press.
8. Brown, A.W. (2012). Enterprise Software Agility: Bringing Agility and Efficiency to the Global Software Supply Chain. Pearson Education.
9. Ambler, S.W. (2013). Surveys Exploring the Current State of Information Technology Practices. <http://www.ambysoft.com/surveys/> Retrieved Jan 4, 2013.

# Security Architecture Approaches

**Sarah Pramanik, Northrop Grumman**

**Abstract.** There are multiple approaches to capturing security architecture information. Each framework and taxonomy has strengths and weaknesses that lend themselves to different tasks. The purpose of this paper is to provide information on some approaches with suggested improvements. The choice of architecture methodology can increase efficiency or diminish it. It is necessary to understand which methodology provides the most effective approach.

## 1. Introduction

Security architecture development is key in determining whether a system will have an adequate security posture. A good security architecture allows system security engineers to understand what mitigations are necessary and how they need to be placed into a system. It also allows for an understanding of information flows between elements of the system and flows out of the system. Without this type of information it is not possible to ensure that the risk level of a system has reduced to an adequate level. There are many taxonomies, frameworks and methodologies for creating security architectures, although there is not a standard required in the DoD.

There are several thoughts on approaching security architectures. The first approach is through adding security into a normal Enterprise Level architecture, which is most common in the DoD, due to the required use of the Department of Defense Architecture Framework (DoDAF) [1]. The second is to have a separate system security architecture [2]. Many of the papers on security architectures [3] define a specific architecture for a specific situation, and are not meant as a pattern on which to base the creation of a system security architecture. These approaches are difficult to work with in designing tactical systems. Designing security architectures for an unmanned aerial vehicle or a submarine is quite different than building security architectures for an office environment.

Even though the requirement sets are identical in some cases, the interpretation and resulting implementation may look radically different. In the Department of Defense there are a few documents that cover security requirements and frameworks, such as the DoD Instruction 8500.2 [4], Information Assurance Technical Framework (IATF) [5], and NIST SP 800-37 [6]. Each system will be designed according to a specific set of security requirements, but the requirement sets are the same whether it is an enterprise type system or a tactical system. The security engineer must understand the purpose of the requirement and the underlying security principles well enough to translate them into a security architecture that will protect unusual systems.

This paper will compare various methodologies commonly used to create security architectures and will provide suggestions on how to apply them for tactical systems. Many of the existing methodologies provide a solid basis for beginning an architecture, although others are sorely lacking. Choosing which methodology to follow is crucial, as it will be the basis on which a security architect will potentially have to work with for many years. The wrong framework can lead to a great headache for engineers and can lead to poor security postures.

## 2. Background

There are multiple frameworks, and taxonomies for the creation of system architectures. Some of these are specific to system security architectures and others are system oriented with security views. In either case, the system security architecture must be totally aligned with the system architecture. This can be especially difficult when designing a non-enterprise type system.

The security architecture is a detailed explanation of how the requirements need to fit within the system. This needs to be done in such a way as to ensure that all vulnerabilities are covered, and not just by one layer of defense, hence the heavy use of the term “defense in depth”. Many methods have been suggested [7, 8, 9, 10, 11, 12, 1, 13, 14, and 15]. Each has a similar goal: build security into the system as a part of the architecture. Many times security is bolted onto the end of system development, or as an upgrade after the system is completed. Each framework and taxonomy has a different way of representing the security elements of a system. One of the earliest taxonomies developed for building enterprise architectures is the Zachman Framework.

The Zachman Framework is used for organizing architectural artifacts [13]. This framework did not supply a formal methodology or process for collecting the artifacts, and so other frameworks were created from this base. The DoDAF was built off of this Framework. SALSA [9] and the Sherwood Applied Business Security Architecture (SABSA) [15] are both built off of the Zachman Framework as well. SABSA is one of the few frameworks developed to focus on the security aspects of a system. Its goal is different than that of the NIST Risk Management Framework (RMF). The intent of the RMF is to improve information security, strengthen risk management processes, and encourage reciprocity among federal agencies. Each of the Frameworks and Taxonomies has strengths and weaknesses.

The biggest complaint made in previous research is that security is done in an ad hoc manner [16]. If an overall security picture of the system is not developed in the beginning, technologies and procedures will be thrown at the system in hopes that something will stick. That leads to the hope that the stuff that sticks is good enough. How can you know what is good enough, without a strategic understanding of the system, its environment, its operations, users and the like. Simply, you cannot. It is not possible to do a risk assessment of a system, if all of the information flows, vulnerabilities, threats and mitigations are not known. The frameworks, taxonomies and methodologies all aim to layout this information.

## 2.1 DoDAF

The DoD requires the use of DoDAF for functional system architectures. The goal of the functional architecture is to show how different types of data should move through the various functions of the system. From this model the system engineers are to derive the interface requirements. This model is also supposed to help define the subsystems and allocation of requirements.

In looking at the DoDAF, each view takes a system and tries to explain it in a different fashion from the other views. There are operational views, system views, technical views, and all views [1]. These describe a system using specific techniques.

The operational views (OV) provide a very high level view. They pictorially depict how different pieces of the system should communicate. These flows do not show any component level information. They show the flows between various organizations, specifically activities, rules, states, events and interoperability flows.

The system views (SV) start to breakdown the flows into system level components. The flows show services, interfaces and data exchanges. This is also the view that is supposed to start to break down the system into a physical schema.

These views do not always show actual protocols. This means that if one segment is planning on using TCP/IP for connection and another is planning on UDP it will not necessarily be evident in this view. It truly depends on how an organization chooses to use the framework. This can lead to confusion between segments. Many times these details are left up to the individual designers of each segment. If there is no communication between the designers then the segments will not communicate.

The technical views (TV) explain the technology standards that should be used in the system, and the emerging standards. These can be difficult to pin down. A system cannot be on contract for an incomplete or unsigned standard. In a system where the development will take several years to complete, it is unknown when the project begins which draft documents will become standards. Typically, this means that if a program lasts for any substantial amount of time then the system will not be created using the newest standards, just those that are on contract.

The all views provide summary information and a glossary. These are helpful in defining the language in which the developers will use to communicate. This can be one of the most useful tools in the framework for developers. The system can only be created if all the designers, developers and architects can communicate clearly. The use of a system wide dictionary can help facilitate this communication.

In each of the views, it is possible to add security attributes. However, this does not adequately express all the security considerations. Security issues can easily be lost in the overall enterprise architecture. This is one of the reasons that, for some systems, it becomes important to break out the system security architecture out of the enterprise architecture and bring it into its own architecture. This also must be done carefully, to ensure that the security architecture aligns with the system architecture.

Another aspect that must be considered in an upcoming program is that DoD is moving from DIACAP to the RMF. This then means that a program will potentially be required to create security views in the DoDAF and follow the NIST RMF. Ensuring that these two methods do not disagree with each other and ensuring that inefficiencies are not increased will be a new task for the system security architects.

## 2.2 NIST RMF

The RMF is designed to allow security engineers to understand the risk level associated with a system. It is supposed to work in conjunction with normal systems engineering. The RMF in conjunction with the allocated information assurance controls from [6] provide something similar to the DoDI 8500.2 [4] and DoDI 8510.01 [17]. The wording in [6] is less vague than that in [4] and provides a more granular way of allocating controls. In and of itself it does not provide guidance on what the security architecture is to look like, nor what it should contain, and is similar to suggestions made in [18]. Its only requirement is that the controls are allocated in such a way as to reduce risk to an acceptable level. It is still up to the security architect to select a method of describing the architecture.

The RMF provides good guidance on boundary determinations as well as words of caution in dealing with SOAs [19]. It also provides a list of criteria that a system security architect should consider when drawing boundary lines and with respect to the overall risk of a system, as well as other information that security engineers would do well to heed. It is also meant to be cyclical.

Once controls are allocated, security engineers are to continually monitor how well they abate the risk and modify mitigations when necessary. Although this is a noble goal, it becomes more challenging if a security architect has not laid out the information in a way that is clear and understandable.

## 2.3 SABSA

The SABSA is a matrix based taxonomy very similar to the Zachman Framework, but with a security twist. Each layer of the SABSA matrix [20] is meant to force the security architect to ask questions to ensure they understand all the various attributes of security mitigations. Each layer takes a different view of the system and provides an explanation to a different set of developers and designers. It is also meant to provide a basis for conversation between the security team and all the other engineers. Although this model provides a good framework for asking questions, it does not match the model for DoD systems.

For example, in looking at the SABSA matrix, the facility manager's view is primarily focused on backups and handling support for what is supposed to be a fixed facility. Although this is a good start, there is a need to tweak this for DoD applications. Physical security is also something that must be kept in mind if dealing with both fixed and mobile platforms. Although physical security is discussed in [2] it assumes fixed facilities, in the DoD environment, the guns, guards, gates and dogs may not always be available to protect a mobile platform such as an Unmanned Aerial Vehicle (UAV). The issue of mobile and fixed platform must not only be part of the CONOPS and functional/conceptual architecture, but must also be folded into the detailed architectures as well.

## 2.4 IATF

The IATF is very different from the RMF and other frameworks. It is a living document that contains information on how to use specific security mitigations such as cross domain solutions and firewalls. There are elements of the document that are not complete and leave room for new information as technology changes.

The IATF is best used in conjunction with the other frameworks to provide the security engineer better information on a particular mitigation than is provided in any of the other frameworks. In and of itself the IATF does not provide a discussion on how to go about putting a security architecture together, but provides useful information on various pieces that might be incorporated into an architecture.

## 3. Tasks

The initial task in creating a security architecture is outlining communication between components within a system and in and out of the system boundaries. This is true whether a system is an enterprise or a tactical system. This outline is similar to the system views in DoDAF, however the security architecture must define which of the flows will be classified (and what levels), sensitive or unclassified. Although the DoDAF allows for security attributes, these do not furnish enough information for the security engineers. This piece of the architecture will be built upon throughout the security architecture process.

Along with this, the security architect must understand the CONOPS and the intended use of each part of the system. This will outline the high level data flows for the system. There is a distressing tendency for system engineers to just throw all the security attributes into one function in the functional architecture. This is especially true if the security engineers are not part of the functional architecture team, or if security is bolted on at the end.

One of the unique aspects of tactical systems, unlike enterprise systems, is that portions of the system may be mobile. If working with an UAV, the architect must take into account not only the permanent ground station, but the UAV and in some cases mobile ground stations. The system may be connecting through differing networks at various classification levels. If the system is going through unprotected territory, there must be means of protecting that information, such as cryptography or the use of volatile memory. This flows both from the understanding of the flow of information as well as CONOPS.

The security architecture needs to have both functional and physical details which must meet the operational needs of the mission. These need to be tied together to ensure that the physical manifestation of the system performs the functions that are intended. These should include such things as key assets, the most critical pieces of information that need to be protected, user interfaces, unsafe environments, etc. Once these are identified, then the next step is to figure out how to mitigate or eliminate these vulnerabilities. This is described in the following approach.

### 3.1 A Modified Approach

The first step in creating the security architecture is to understand the requirements of the system. Aside from the information assurance requirements, such as the DoDI 8500.2 [4], the concept of operations, data classification and data exchanges must be recognized. This information should be translated into a series of block diagrams. Although a functional architecture must exist showing this information, it will not necessarily have to overlay on a physical diagram at the end. One thing to note is that as soon as the diagram goes under configuration control, it makes it difficult to maintain.

The first diagram should show internal and external flows of information similar to the one shown in Figure 1. The DoDAF OV-1s provide similar information from a functional standpoint. The security architect should ensure that the functional information matches the physical architecture from a security standpoint. The complexity of this view can increase significantly when dealing with multiple levels of security as opposed to operating a single security level.

Once the basic flows are understood, the classification level of each flow should be identified. There should also be notations of where data will cross a classification boundary, such as shown in Figure 2. This information allows the security architect to know where specific solutions such as encryption, cross domain solutions or data guards might need to be employed.

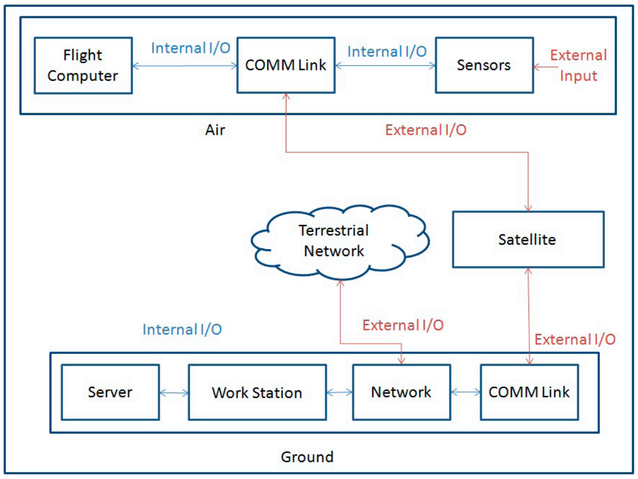


*Figure 1. Flows of the System*

As the DoDAF functional architecture is being completed, a corresponding physical architecture is typically being created, especially when the program is employing concurrent engineering. The physical architecture should be of great interest to the security architect. As the physical architecture is being designed, it will go through multiple iterations.

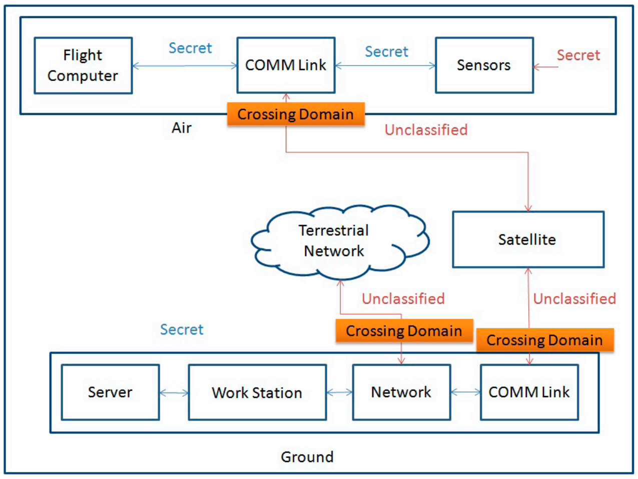
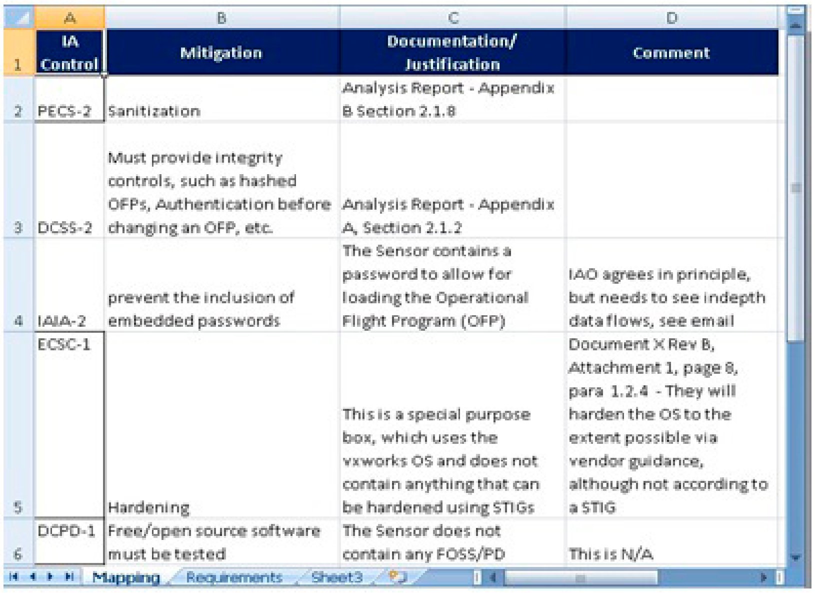


*Figure 2. Classification Domains*

As the first physical architecture is laid out, it becomes a base on which to build the physical security architecture. One of the primary purposes of the security architecture is to ensure that the security requirements are being met in the system. As subsystems and components are being defined the security engineers should begin listing the security requirements that will be applicable to each item. A spreadsheet can be maintained for each major component, in the beginning.

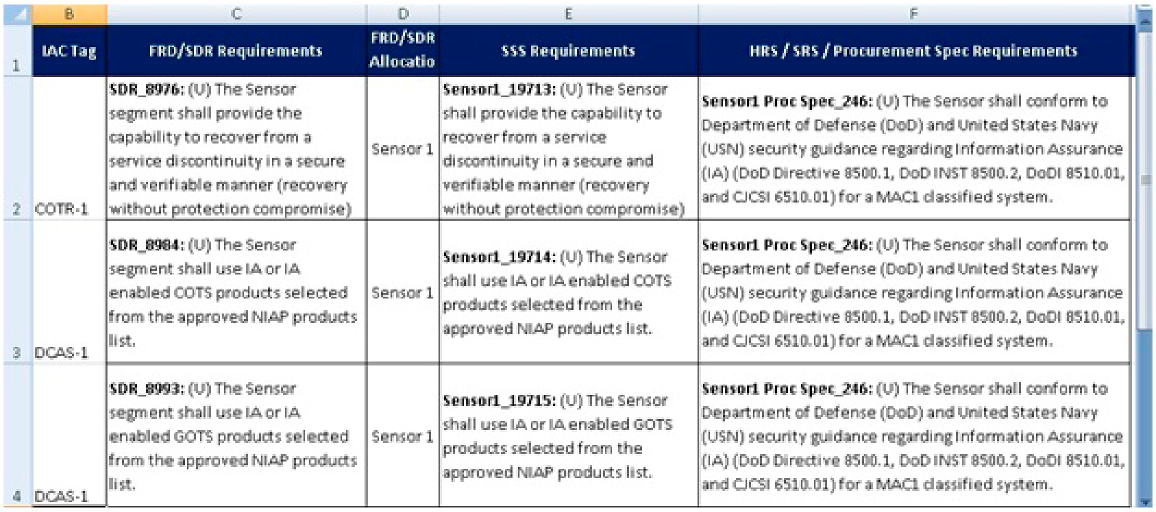
The first tab should map the IA controls to the parts of the component and to the expected mitigation, as seen in Figure 3. The engineer can include notes and justification for clarification. The second tab should map the IA control to the requirements associated with the control, as seen in Figure 4. The IA Control is used to link the mapping and the requirement tabs together.



| | IA Control | Mitigation | Documentation/ Justification | Comment |
|---|---|---|---|---|
| 2 | PECS-2 | Sanitization | Analysis Report - Appendix B Section 2.1.8 | |
| 3 | DCSS-2 | Must provide integrity controls, such as hashed OFPs, Authentication before changing an OFP, etc. | Analysis Report - Appendix A, Section 2.1.2 | |
| 4 | IAIA-2 | prevent the inclusion of embedded passwords | The Sensor contains a password to allow for loading the Operational Flight Program (OFP) | IAO agrees in principle, but needs to see indepth data flows, see email |
| 5 | ECSC-1 | Hardening | This is a special purpose box, which uses the vxworks OS and does not contain anything that can be hardened using STIGs | Document X Rev B, Attachment 1, page 8, para 1.2.4 - They will harden the OS to the extent possible via vendor guidance, although not according to a STIG |
| 6 | DCPD-1 | Free/open source software must be tested | The Sensor does not contain any FOSS/PD | This is N/A |

*Figure 3. IA Control Mapping*

This mapping allows for the security engineers to double check that requirement wording has been allocated appropriately. In the example, the security engineer should see this and note a need for a change in procurement specification wording. This is also an important part of the overall system security engineering method, because not all aspects of the security requirements can be laid out on a physical diagram. Requirements such as configuration management must be met for every component, but are more accurately addressed using language rather than diagrams. Later on in the lifecycle, the mapping can also be used to link it to verification of each of the controls.



| | IAC Tag | FRD/SDR Requirements | FRD/SDR Allocatio | SSS Requirements | HRS / SRS / Procurement Spec Requirements |
|---|---|---|---|---|---|
| 2 | COTR-1 | **SDR_8976:** (U) The Sensor segment shall provide the capability to recover from a service discontinuity in a secure and verifiable manner (recovery without protection compromise) | Sensor 1 | **Sensor1_19713:** (U) The Sensor shall provide the capability to recover from a service discontinuity in a secure and verifiable manner (recovery without protection compromise) | **Sensor1 Proc Spec_246:** (U) The Sensor shall conform to Department of Defense (DoD) and United States Navy (USN) security guidance regarding Information Assurance (IA) (DoD Directive 8500.1, DoD INST 8500.2, DoDI 8510.01, and CJCSI 6510.01) for a MAC1 classified system. |
| 3 | DCAS-1 | **SDR_8984:** (U) The Sensor segment shall use IA or IA enabled COTS products selected from the approved NIAP products list. | Sensor 1 | **Sensor1_19714:** (U) The Sensor shall use IA or IA enabled COTS products selected from the approved NIAP products list. | **Sensor1 Proc Spec_246:** (U) The Sensor shall conform to Department of Defense (DoD) and United States Navy (USN) security guidance regarding Information Assurance (IA) (DoD Directive 8500.1, DoD INST 8500.2, DoDI 8510.01, and CJCSI 6510.01) for a MAC1 classified system. |
| 4 | DCAS-1 | **SDR_8993:** (U) The Sensor segment shall use IA or IA enabled GOTS products selected from the approved NIAP products list. | Sensor 1 | **Sensor1_19715:** (U) The Sensor shall use IA or IA enabled GOTS products selected from the approved NIAP products list. | **Sensor1 Proc Spec_246:** (U) The Sensor shall conform to Department of Defense (DoD) and United States Navy (USN) security guidance regarding Information Assurance (IA) (DoD Directive 8500.1, DoD INST 8500.2, DoDI 8510.01, and CJCSI 6510.01) for a MAC1 classified system. |

*Figure 4. Requirement Mapping*

Dynamic Object-Oriented Requirements System or a similar requirement repository under configuration control should be used to maintain the information once it has reached a baseline. The mappings will change as the security architecture matures, but in order to begin creating the security views of the system, the security architect must first understand the basic function and contribution of each subsystem and component.

There are IA concepts to consider in DoD systems, regardless of the IA requirement set. These can be defined by the security views as listed in Table 1 and will change over time, as components are added or changed in the system. Also, there are times when control of a subsystem function may move from one integrated product team to another. For example, boundary defense may originally be allocated to both the communications IPT and the ground segment IPT. As the system grows, the network piece may then be owned by the communications IPT. In the example above, this then would mean that the communications IPT now owns the responsibility for all boundary defense in the system. The requirement allocation would stay the same, and the diagram would stay the same, but the ground segment would claim inheritance for meeting boundary defense. This would then be added back into the requirements mapping discussed earlier.

Each view is to be overlaid onto the physical architecture. By layering these views, using a tool such as Microsoft™ Visio, the defense in depth picture can be seen. It also allows the architect to see where vulnerabilities may still exist. The earlier in the life cycle of a program that an accurate architecture can be created, the easier it is for security engineers to develop affordable security mitigations.

In some cases there may be multiple physical architectures. For example, if an unmanned vehicle and a ground station are being designed, creation of two physical architectures is necessary. They must show the interconnections between the two

| **Security View** | **Describes** |
|---|---|
| Domains and Interconnections | - Interconnections with other enclaves or networks<br>- Accreditation boundaries<br>- Should show the classifications of data within the system<br>- Points of red/black separation with respect to TEMPEST boundaries |
| Network Flow | - Boundary Defense<br>- Data at Rest (indicate protection)<br>- Data in Transit (indicate protection)<br>- Network management data<br>- Partitioning of User Interfaces and Storage<br>- VoIP/Phone lines (indicate protection) |
| Cryptographic Usage | - Certificate Authority<br>- Key Storage<br>o Asymmetric<br>o Symmetric<br>- Interfaces to Electronic Key Management System (EKMS)<br>- DS-101/102 lines<br>- Key loading areas (at the box, or a harness, etc.)<br>- Key zeroization paths (hardware, software paths)<br>- Cryptographic type: Type-1, Type-2, etc.<br>o Embedded, separate hardware, etc. |
| Handoff (for unmanned systems) | - Outline control steps<br>o How control is acquired<br>o How control is passed<br>o Include authentication steps<br>o *Non-repudiation of Command and Control items are critical and should be indicated |
| Patch Management | - Indicate which pieces of equipment have software/firmware that will require patching<br>- Indicate how patches will be downloaded, tested, protected and applied<br>- *It should be noted that patches will require regression testing of system elements |
| Mobile Code | - Indicate pieces of the system potentially have access to mobile code<br>- Level of mobile code<br>- Protections against unwanted mobile code |
| Audit | - Indicate elements of the system with audit capability<br>- What is audited(e.g. can only log user, but not time stamp)<br>- Central audit repository |
| Backup/Recovery | - Location of recovery software( e.g. backup images in safe)<br>- Location of data that has to be backed up<br>- Location of backup repository<br>- Timing of backups (weekly, daily, etc.)<br>- *This provides useful information for the disaster recovery plan |
| Zeroization/<br><br>Sanitization | - Define memory element in every component, (volatile, non-volatile)<br>- Type of memory<br>- Classification of data in each memory type<br>- Sanitization approach for NVM<br>- Zeroization approach for Volatile memory<br>- Associated analysis for components that plug into classified networks, but are claiming to remain unclassified at shutdown |
| System State Changes | - Identify when the system is considered classified, unclassified (or range of classification)<br>- Should have a view for system running, powered off, and in-transit (if applicable) |
| IA Physical Security (Typically associated with facilities requirements) | - Status of platform (mobile, fixed, semi-fixed, etc.)<br>- Fire suppression systems (type and placement)<br>- Smoke detectors (placement and type)<br>- Temperature/humidity controls (for humans and equipment)<br>- Lighting (emergency)<br>- Power (backup generators, failover period, etc)<br>- *Depending on the system, these items may already be in place and a site survey will provide this data |
| Other Physical Security | If a new building is being constructed, a separate physical security architecture will have to be created. These items are not generally covered in IA documentation. For reference see [21]. |
| Authentication | - Indicate how/where users will authenticate to the system<br>o Single sign on, role based access control, group accounts, etc<br>- Indicate machine to machine authentication<br>o Note how the authentication is being achieved |

*Table 1. Security Views*

pieces and any interconnections to outside networks that may be used to allow them to communicate. Developing security views for both architectures is also essential.

One of the difficulties is ensuring that as the physical architecture changes, the security architecture changes with it. This is especially true when the network diagrams and ports, protocols and services are in the process of being defined. These flows must be captured as part of the network view. They will determine how access control lists and firewall settings are configured in the final system.

Creating a security architecture for DoD systems requires the architect to be in line with both the information assurance control allocation and the required certification and accreditation process. This means that the architecture must show how the controls have been allocated, why they have been allocated and how it affects the system. This must be documented in a way that allows the security architect to provide justification for allocation of controls as well as an understanding of the security posture of the system if a control is not met or something in the system changes. This is fundamental to ensuring the security posture of the system remains at an acceptable level.

If the security architect defines all of the above views and maintains the requirements mappings, the evidence necessary for certification and accreditation is also available. The vulnerabilities in the system can be examined and either mitigated or justified. This approach uses a similar process as SABSA, but with a tactical focus and not an enterprise focus.

## 4. Conclusion

The use of the appropriate security architecture methodology, taxonomy or framework defines the degree of difficulty an architect will have in adequately describing the security functionality of a system. The modified approach provided in this paper allows for incorporating the tactical nature of many DoD systems into the equation. If followed, it also ensures that the security requirements are accounted for. This is critical to achieving certification and accreditation.

## ABOUT THE AUTHOR

Sarah Pramanik is the lead for software information assurance (IA) on her current program. She is responsible for leading the development, integration and execution of IA activities within the software team. Sarah has worked in multiple areas of cyber security, including vulnerability and penetration testing, information system security engineering, cyber security training and security architecting. Sarah holds a Bachelors and Masters degree in Computer Science from The University of Colorado, Colorado Springs.

**Northrop Grumman Aerospace Sector**
**600 Grumman Road West M/S Z21-025**
**Bethpage, NY 11714**
**Phone: 516-346-7737**
**E-mail: Sarah.Pramanik@ngc.com**

## REFERENCES

1. Wikipedia. Department of Defense Architecture Framework. 14 December 2010. <http://en.wikipedia.org/wiki/Department_of_Defense_Architecture_Framework>.
2. John Sherwood, et al. Enterprise Security Architecture: A Business-Driven Approach. San Francisco, CA: CMP Books, 2005.
3. Elkins, Asa, Jeffery Wilson and Denis Gracanin. "Security Issues in High Level Architecture Based Distributed Simulation." Proceedings of the 2001 Winter Simulation Conference. 2001 © WSC'01.
4. Department of Defense. "Department of Defense Instruction 8500.2: Information Assurance (IA) Implementation." February 6, 2003.
5. National Security Agency Information Assurance Solutions Technical Directors. "Information Assurance Technical Framework, Release 3.0." September 2000.
6. National Institute for Standards and Technology Special Publication 800-37 R1. "Guide for Applying the Risk Management Framework to Federal Information Systems: A Security Life Cycle Approach." Guide for Applying the Risk Management Framework to Federal Information Systems: A Security Life Cycle Approach. Gaithersbergm MD 20899-8930: Computer Security Division Information Technology Laboratory National Institute of Standars and Technology, February 2010.
7. Blackwell, Clive. "A Multi-layered Security Architecture for Modelling Complex Systems." CSIIRW'08 May 12-14. 2008 © ACM.
8. Susanto, Heru. and Fahad bin Muhaya. "Multimedia Information Security Architecture Framework." 2010 © IEEE.
9. Sherwood, John. "SALSA: A Method for Developing the Enterprise Security Architecture and Strategy." 1996 © Published by Elsevier Science Ltd. doi:10.1016/S0167-4048(97)83124-0, n.d.
10. Neilforoshan, Mohamad. "Network Security Architecture." © Consortium for Computing Sciences in Colleges, 2004.
11. National Information Assurance Glossary. "CNSS Instruction No. 4009." 26 April 2010.
12. Oda, Michelle, Huirong Fu and Ye Zhu. "Enterprise Information Security Architecture A Review of Frameworks, Methodology and Case Studies." 2009 © IEEE.
13. Wikipedia. Zachman Framework. 10 December 2010. <http://en.wikipedia.org/wiki/Zachman_Framework>.
14. Wikipedia. "Information Security." 10 December 2010. <http://en.wikipedia.org/wiki/Information_security>.
15. SABSA Ltd. The SABSA Method. n.d. <http://www.sabsa-institute.org/>.
16. Hartig, Hermann. "Security Architectures Revisited." EW 10 Proc. 10th Workshop on ACM SIGOPS European workshop. 2002 © ACM.
17. Department of Defense. "Department of Defense Instruction 8510.01: DoD Information Assurance Certification and Accreditation Process (DIACAP)." November 28, 2007.
18. Suhair Hafez Amer and John A. Hamilton, Jr. "Understanding Security Architecture." SpringSim (2008).
19. National Institute of Standards and Technology . "Special Publication 800-53 Revision 3, August 2009, includes updates as of 05-01-2010" Recommended Security Controls for Federal Information Systems and Organizations." Gaithersburg, MD, August 2009.
20. SABSA Ltd. 2010 ©SABSA. The SABSA Matrix. n.d. <http://www.sabsa.org/the-sabsa-method/the-sabsa-matrix.aspx>.
21. Department of the Army. "FM 3-19.30: Physical Security." 8 January 2001.

# Secure DoD Software

## Considerations for the Vulnerability Market

**Major Bradley C. Panton, Air Force Institute of Technology**
**John M. Colombi, Ph.D., Air Force Institute of Technology**
**Michael R. Grimaila, Ph.D., Air Force Institute of Technology**
**Robert F. Mills, Ph.D., Air Force Institute of Technology**

**Abstract.** Every year, the DoD upgrades their information technology systems, allows new applications to connect to the network, and reconfigures the Enterprise to gain efficiencies. While these actions are to better support the warfighter and satisfy national security interests, they introduce new system vulnerabilities waiting to be exploited. This article recommends the DoD enter the vulnerability marketplace to mitigate the risk of a cyber attack using these undiscovered vulnerabilities. Through use of the vulnerability market, DoD will ensure information security is built into the application, minimize the number of distributed patches, and optimize investment in defense programs.

The vulnerability market, otherwise known as the market for “zero-day” vulnerabilities, has thrived ever since the first exploit was discovered on a computer system. Starting out as a black market forum where hackers could trade information for money, the vulnerability market is transitioning to a legitimate service. The vulnerability market now has growing influence over DoD software developers who regard computer security as a critical and required capability, and not just an added feature.

Historically in the DoD, as budgets contract, information systems aggregate. This phenomenon occurs primarily to offset the expense of maintaining a large workforce by automating much of the work accomplished by soldiers, sailors, airmen, and marines. As a consequence, an increase in the number of automated processes drives an increase in the number and complexity of information systems. The negative externality associated with this phenomenon is that as the number, complexity, and size of information systems increase, the prevalence of system flaws also increase. For example, a 2010 RAND study reported that a typical large code base can have a rate of one defect for every thousand software lines of code (KSLOC). Applying this defect rate to the Joint Strike Fighter’s 18,000 KSLOC, there may be as many as 18,000 defects. While only a fraction of these defects would allow access to the IS and lead to unauthorized control of the system, an entirely defect-free information system is realistically impossible to achieve.

In order to mitigate the release of a system with undiscovered vulnerabilities, the DoD acquisitions process goes through great lengths to test the security of a product. Through developmental and operational test and evaluation, penetration testing, and the comprehensive information assurance certification and accreditation Process, the DoD seeks to identify and mitigate the risk of a possible cyber attacks resulting in the loss of money and life. These tests, coupled with the bolted on defense-in-depth strategy, have one critical shortfall; none of them analyze the system for undiscovered or obscure vulnerabilities.

The vulnerability disclosure lifecycle of a system typically consists of three common phases: learning, linear, and saturation [1], as shown in Figure 1. These phases are important as vulnerability discovery rates increase and decrease over time as the system passes through each window. The learning phase occurs immediately after the system is released to the public. During this phase, researchers and hackers become familiar with the system and gain better knowledge on how to break it. As a result of this lack of system knowledge, the vulnerability discovery rate during this phase tends to be low. Following the learning phase, the linear phase is characterized by a linear growth of vulnerabilities discovered by users. This explosion of discoveries is due to the system gaining market penetration and an increase in system familiarity. Once the system reaches obsolescence or as the number of undiscovered vulnerabilities diminishes, the vulnerability rate reduces as more users convert to a replacement and hackers lose interest. During this time the system is experiencing the saturation phase.

The length of time a system experiences each of the phases varies greatly. For example, if the hackers adapt to the new system quickly, the learning phase is short-lived. Furthermore, if the system is rife with vulnerabilities, the saturation phase may never be seen. Examples of these phases are readily seen in the commercial market. For demonstrative purposes, three popular systems are shown in Figure 2: Adobe Acrobat, the Java Development Kit (JDK), and Windows XP.

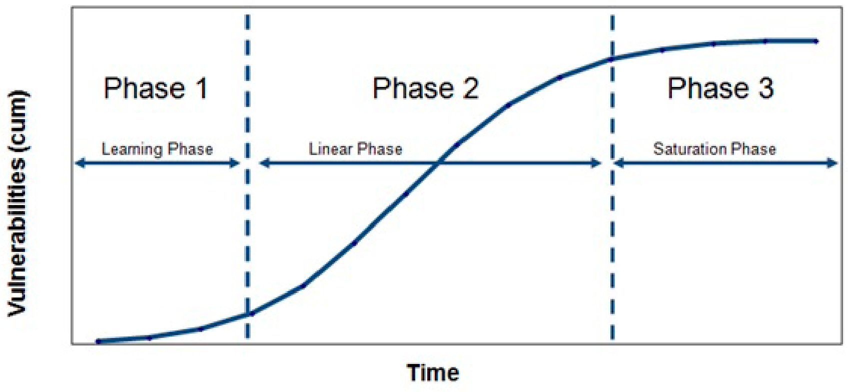


*Figure 1: Vulnerability Disclosure Rate Phases [1]*

As shown in Figure 2, there are clear delineations between the learning and linear phases. Also of note is the variability of phase lengths between software systems. Windows XP’s learning phase was approximately three years where Adobe Acrobat experienced a 10-year learning phase. The causal factor of this variability is based on market share. For the Windows XP operating system, consumers quickly upgraded from the obsolete

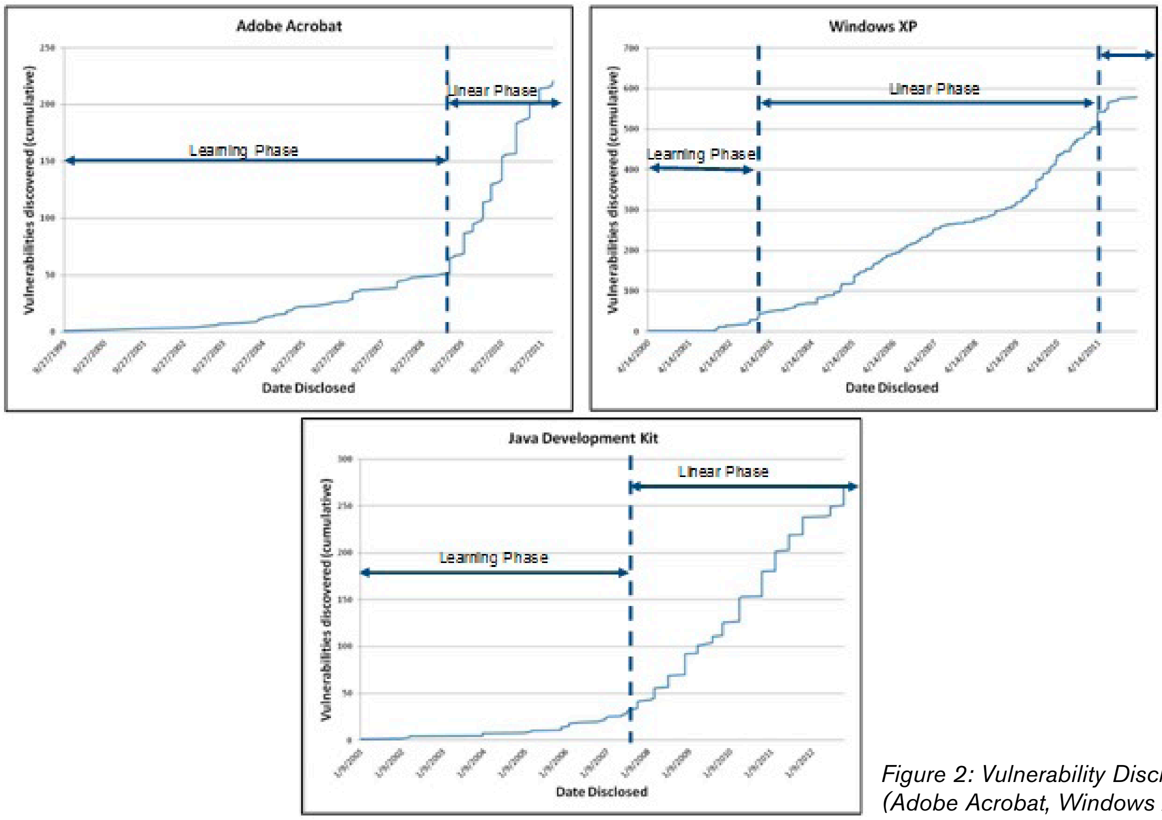


*Figure 2: Vulnerability Disclosure Histories (Adobe Acrobat, Windows XP, JDK)*

Windows 98/NT systems. The quick conversion ensured that Windows XP gained a large share of the market over a relatively short amount of time. In contrast, the Adobe Acrobat's share of the Portable Document Format market was limited by competitor saturation. It wasn't until July 2003 and the release of Adobe version 6.0 that the system gained popularity over similar proprietary systems. Shortly after the 2003 release, Adobe Acrobat entered the linear phase.

While the Common Vulnerabilities and Exposures database allows historical trend analysis, researchers have been searching for a model that will allow for predictive study. One such model is the Alhazmi-Malaiya Logistic (AML) model [1]. The AML model assumes that the shape of the vulnerability curve is restricted by market share and the number of the undiscovered vulnerabilities. The model proposes that the vulnerability discovery rate is given by the differential equation, Equation 1:

$$\frac{d\Omega}{dt} = \mathrm{A}\Omega\,(B - \Omega)$$

*Equation 1:*

The two factors in Equation 1, AΩ and (B – Ω), relate to the application's market share and the number of system vulnerabilities. AΩ increases as market share increases and (B – Ω) decreases as the number of available vulnerabilities (B) decrease. Solving for Ω(t), the following logarithmic equation, Equation 2, is produced:

$$\Omega(\mathrm{t}) = \frac{\mathrm{B}}{BCe^{-ABt} + 1}$$

*Equation 2:*

In this equation, as time (t) approaches infinity, Ω(t) approaches B. Assuming the other variables remain constant, decreasing the number of vulnerabilities in a system (B) would flatten the shape of the s-curve. Stating that the market share (AΩ) remains constant is appropriate for DoD. More often than not, DoD acquires a specific application or system to meet a specified mission. Consequently, that system has a constant market share within the DoD. As a DoD system becomes obsolete and replaced, there is a resultant transition time; however, it has an accelerated pace which limits the saturation phase. As noted before, the delivery of a defect-free information system is impossible to achieve. The DoD can, however, attempt to deliver a system that is void of as many defects as possible, prior to deployment to the warfighter and operational use.

How does the DoD calculate the cost of a cyber attack? This question is not easily answered as there are many factors that determine total cost. In 2011, a global network security powerhouse, McAfee, reported the global economic impact to cyber attacks is as large as $1 trillion dollars. Furthermore, General Keith Alexander, commander of USCYBERCOM and Director of the NSA, estimated that the U.S. loses $250 billion annually to cybercriminals [2]. While a detailed account on how these estimates were formulated is not available, the public can assume the estimates were built using the following categories:

- Costs in anticipation of a cyber attack. Include the DoD's investment in the cyber security architecture (such as installing and implementing the Defense-in-Depth strategy).
- Costs as a consequence of a cyber attack. Takes into account the direct losses to an individual, service, defense industrial base, and overall national security.
- Indirect costs associated with a cyber attack. Includes damage to an organization's reputation, loss in national confidence, and time required to recover [3].

In the civilian sector, costs can be enumerated by the number of credit card numbers stolen, intellectual property theft, and pilfered insider trading information. In the defense sector, costs are measured as impacts to operations and intelligence activities. Based on the complexity of devising costs for cyber attacks, this article generalizes "cost" by calculating a probabilistic outcome using expected values.

In an effort to identify how the vulnerability market can strengthen overall system security, some basic formulas used to model the risk of a system to a particular vulnerability will be defined. For this analysis, we use the Single Loss Expectancy (SLE) formula to calculate the expected loss due to an exploited vulnerability. The SLE calculates a value based on the occurrence of a risk on a system. Calculating the SLE for a system incorporates two factors: the value of the at-risk asset (AV) and the asset's Exposure Factor (EF). The EF is a percentage of the asset's value that will be lost in the case of an attack. In the DoD, quantifying AV is difficult as it includes the value of information, value of lost productivity, the value of remediation, and (in extreme cases) the value of human life.

Suppose the DoD has an information technology asset (A) that is vulnerable to a particular system vulnerability (j). Let AV be the value of A and let EFj be the exposure factor for asset A when A is successfully attacked through the vulnerability j. Furthermore, let Pj be the probability of a successful attack on A through the vulnerability j. By incorporating these variables, the SLE for a successful attack results in Equation 3:

$$\textit{Single Loss Expectancy (SLE)} = (AV \times EF_j) \times P_j$$

*Equation 3:*

The resultant SLE value is the cost risk that the organization incurs by not mitigating the probability of a particular vulnerability being exploited. Assuming an asset's value remains constant, the SLE can be reduced by either lowering the exposure factor or the probability of a successful attack.

It is unrealistic to believe a system in the DoD inventory is only susceptible to a single vulnerability. In fact, a DoD system may have hundreds of unknown vulnerabilities. To account for the entire set of vulnerabilities against a particular system, the Total Expected Loss for the set of all possible vulnerabilities {Tj} is the summation of SLEs. The sum of system SLEs, or Total Expected Loss (TEL), is expressed using Equation 4:

$$\textit{Total Expected Loss } (TEL) = \sum_{j=1}^{n} SLE_j = \sum_{j=1}^{n} (AV \times EF_j) \times P_j$$

*Equation 4:*

For a given system, there are a total of n vulnerabilities. Now assume that the DoD engages in a strategy in which a set of vulnerabilities {Uj} are identified with set {Uj} being a subset of all possible {Tj}. By integrating this set of identified vulnerabilities, the new total expected loss (TEL') Equation 5 is:

$$TEL' = \left[\sum_{j \in T_j}^{n} (AV \times EF_j) \times P_j\right] - \left[\sum_{j \in U_j}^{n} (AV \times EF_j) \times P_j\right] - \sum_{j \in U_j}^{n} Price_j$$

*Equation 5:*

This set of identified vulnerabilities {Uj} effectively removes each corresponding SLE by changing the probability of attack from Pj to 0. Since {Uj} is a subset of {Tj}, the difference between the two summations is a positive value. As long as the cost of the purchased vulnerabilities (∑Pricej) is less than the difference, the expected net benefit is positive.

In acquiring secure software systems and applications, DoD could incentivize developers to use a mechanism that discovers the set of vulnerability disclosures {Uj} at a fair market price (∑Pricej) as part of development costs. One promising mechanism is the Vulnerability Market.

A vulnerability market is a setting where researchers are rewarded for discovered software vulnerabilities. On May 15th, 2013 the DHS announced that the government is entering the vulnerability marketplace by selling its stockpile of zero-day vulnerabilities to qualified vendors [4]. Furthermore, national media outlets have reported that the NSA actively researches and purchases zero-day exploits in order to gain access to an adversary's cyber assets [5]. While the precedence and legal framework are well established, the DoD has yet to realize the potential value of paying third party researchers for vulnerability information may have on DoD systems. Surprisingly, industry understands the issues of software vulnerability prevalence better than the DoD. In the past decade, dozens of vulnerability markets have sprung into existence based upon the perceived need to enlist non-organic researchers to report application vulnerabilities.

Today, the two primary players in the commercial vulnerability market are iDefense and Hewlett Packard TippingPoint's zero-day initiative. Between March 2003 and December 2007 an average 7.5% of the vulnerabilities affecting Microsoft and Apple were processed by either iDefense or TippingPoint [6]. Since 2007, the CanSecWest security conference has hosted the annual Pwn2Own bug challenge which rewards researchers for hacking into some of the most popular computer applications. During the 2013 Pwn2Own challenge, researchers were awarded $480,000 for cracking applications developed by Microsoft, Google, Adobe, Mozilla, and Oracle. Even more impressive, Google claimed theirs was the most secure operating system on the market by offering $110,000 for a browser or system level compromise delivered via a web page. At the end of the conference, the entire Google prize pot of $3.14M remained intact [7].

Each information system vulnerability has the probabilistic potential to cost the DoD resources. Although calculating the consequences of using a system with unknown vulnerabilities is difficult to quantify, it is certain that the discovery of a vulnerability prior to it being exploited by an adversary is more cost effective than remediating it post attack. Decreasing the probability and increasing the discovery rate of system vulnerabilities is the primary goal of using the vulnerability market for DoD systems. Not only will the discovery of an unknown vulnerability effectively reduce the probability of a successful attack, lifecycle operations, maintenance costs, and remediation efforts will also be reduced.

## ABOUT THE AUTHORS

Major Bradley C. Panton is an acquisitions program manager for the United States Air Force. As a program manager, Major Panton has worked at the Electronic Systems Center, the Missile Defense Agency, and the National Geospatial-Intelligence Agency. He holds degrees in operations research, military operational arts and sciences, and cyber warfare.

**E-mail: bradley.panton@us.af.mil**

Dr. John M. Colombi is an Assistant Professor of Systems Engineering at the Air Force Institute of Technology. He teaches graduate courses and leads sponsored research in support of the Systems Engineering program. Dr. Colombi served at the NSA developing information security and managed communications networking research at the Air Force Research Laboratory.

**E-mail: john.colombi@afit.edu**

Dr. Michael R. Grimaila is an Associate Professor of Systems Engineering and Management and member of the Center for Cyberspace Research (CCR) at the Air Force Institute of Technology (AFIT), Wright-Patterson AFB, Ohio USA. He is a Certified Information Security Manager (CISM), a Certified Information Systems Security Professional (CISSP), a member of the ACM, a Senior Member of the IEEE, and a Fellow of the ISSA.

**E-mail: michael.grimaila@afit.edu**

Dr. Robert F. Mills is an Associate Professor of Electrical Engineering in the Department of Electrical and Computer Engineering, Air Force Institute of Technology. He teaches and conducts research in a variety of areas to include cyber security, network operations and management, electronic warfare, and systems engineering. He serves as the Curriculum Chair for AFIT's Cyber Warfare masters program and is a Senior Member of the IEEE.

**E-mail: robert.mills@afit.edu**

## REFERENCES

1. Younis, A. A., Joh, H., & Malaiya, Y. K. (2011). Modeling Learningless Vulnerability Discovery using a Folded Distribution. The 2011 International Conference on Security and Management , 617-623.
2. Maass, P., & Rajagopalan, M. (2012, August 1). Does Cybercrime Really Cost $1 Trillion? Retrieved May 15, 2013, from Pro Publica: <http://www.propublica.org/article/does-cybercrime-really-cost-1-trillion>
3. Office of Cyber Security and Information Assurance. (2011, February). Retrieved May 17, 2013, from The cost of cyber crime: <http://www.cabinetoffice.gov.uk/resource-library/cost-of-cyber-crime>.
4. Schwartz, M. J. (2013, May 16). DHS Eyes Sharing Zero-Day Intelligence With Businesses. Retrieved May 17, 2013, from Information Week: <http://www.informationweek.com/security/vulnerabilities/dhs-eyes-sharing-zero-day-intelligence>.
5. Schneier, B. (2012, May 30). Forbes. Retrieved May 22, 2013, from The Vulnerabilities Market and the Future of Security: <http://www.forbes.com/sites/bruceschneier/2012/05/30/the-vulnerabilities-market-and-the-future-of-security/>.
6. Frei, S., Schatzmann, D., Pattner, B., & Trammel, B. (2009, June 25). Modeling the Security Ecosystem – The Dynamics of (In)Security. Retrieved May 22, 2013, from <http://www.techzoom.net/security-ecosystem>.
7. Thomson, I. (2013, March 8). Pwn2Own: IE10, Firefox, Chrome, Reader, Java hacks land $500k. Retrieved March 13, 2013, from <http://www.theregister.co.uk/2013/03/08/pwn2own_contest_cansecwest/>.

Homeland Security

The Department of Homeland Security, Office of Cybersecurity and Communications (CS&C) is responsible for enhancing the security, resiliency, and reliability of the Nation's cyber and communications infrastructure and actively engages the public and private sectors as well as international partners to prepare for, prevent, and respond to catastrophic incidents that could degrade or overwhelm these strategic assets. CS&C is seeking dynamic individuals to fill critical positions in:

- Cyber Incident Response
- Cyber Risk and Strategic Analysis
- Networks and Systems Engineering
- Computer and Electronic Engineering
- Digital Forensics
- Telecommunications
- Program Management and Analysis
- Vulnerability Detection and Assessment

**To learn more about the DHS, Office of Cybersecurity and Communications, go to www.dhs.gov/cybercareers. To apply for a vacant position please go to www.usajobs.gov or visit us at www.DHS.gov.**

# It Is Not Too Late for Software Assurance!

**Robert A. Voitle, Jr.,**
**Application Software Assurance Center of Excellence**
**Arthur J. Boote,**
**Application Software Assurance Center of Excellence**
**James "Woody" Woodworth,**
**Application Software Assurance Center of Excellence**

**Abstract.** Software Assurance is the practice of designing secure software that can safely and reliably operate in a hostile environment and resist attacks when all other network defenses have failed. Real-time IA focuses on mitigating attacks while within that hostile environment but can be greatly aided by Software Assurance practices regardless of where an application is in its lifecycle. In 2012, Yahoo suffered from an attack on an application initially developed by another company and had Yahoo executed Software Assurance techniques on the mature application, they could have prevented a compromise that resulted in the release of more than 400,000 user names and passwords.

## Introduction

In May of 2009, President Obama said, "Our technological advantage is a key to America's military dominance. In today's world, acts of terror could come not only from a few extremists in suicide vests but from a few key strokes on the computer—a weapon of mass disruption!"

So, how secure is your application? This is a question we ask each program office before and after Software Assurance Risk Assessments. At the start, programs are very confident with the security of their "baby"; but, once we finish, that confidence is usually reduced. As they discover, the good news is all can be fixed; and, with the proper training, tools, techniques, methods and processes, they can ensure continuous application security from within to couple with the real-time Information Assurance efforts deployed by their hosts.

Software Assurance is the discipline of defensively coding software applications/systems in order to harden them from compromise/hacking. While traditional software engineering and coding practices emphasize user functionality and follow-on sustainability: deferring security to systems-level design), Software Assurance, by comparison, bakes security into the software itself. It does not come at the expense of user functionality or follow-on sustainability—it adds security up-front, in a deliberate effort to make the systems as least-hackable as possible, especially those operating in contested environments.

In addition to being a sound systems engineering practice, Software Assurance is now a legislative requirement. More specifically, Section 933 of Public Law 112-239 [1] requires the Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics and DoD Chief Information Office to develop and implement a baseline Software Assurance policy for the entire lifecycle of systems.

To fully comply with P.L. 112-239 and improve Mission Assurance for the ever-increasing majority of software-enabled systems, enforcement mechanisms must be put in place to require assessments prior to system fielding as well as throughout their lifecycle. Current policy, guidance, enforcement mechanisms and force structures are inadequate for full compliance.

In today's ever-changing environments, Software Assurance is Mission Assurance! Government and industry leaders in the software security community have advocated for years for widespread Software Assurance strategy implementation. They have struggled with insufficient official guidance, the absence of dedicated funding sources and lopsided emphasis on hardware and network security instead of software, where industry analysts believe the majority of attacks occur. Acquiring and building software without accounting for security is no longer an acceptable risk.

The "way ahead" is simple…the DoD must proactively find, identify, and assess weaknesses that may be in software developed and/or used by the warfighter before, during and after fielding. By building security in early and often throughout the software's lifecycle, we tie security into the overall quality and functionality of systems, which not only prevents malicious entities from hacking our systems proactively, but saves millions of dollars per year in cost avoidance caused by flaws in the code, work stoppage from system failure, re-engineering, patching and re-fielding. Software Assurance is not just a "Just-In-Time" process utilized for fielding new technologies but allows us to apply techniques to identify and fix problems due to bloat, years of various coding techniques, and vulnerabilities inherent in Legacy Software, which houses the vast majority of our critical information; thereby hardening the current infrastructure.

## Case Study: Yahoo Voices

In July of 2012, a hacking group posted usernames and passwords that had been pulled from a Yahoo sub-domain. The group itself revealed that their attack, a SQL injection that the attackers used to pass information retrieval commands to Yahoo's database servers, had allowed them to retrieve almost half a million unencrypted email addresses and passwords [2]. As the details of the attack were released to the public, Yahoo's image suffered from both the mistakes that allowed the attack to happen and from its poor damage control actions.

The problems date back before May of 2010, when Yahoo purchased Associated Content for $100 million to rebrand as its Yahoo Voices service. Associated Content provided a site for writers and subject matter experts to have an official platform for their articles and videos [3]. Unfortunately, Associated Content had not built their platform securely and Yahoo had not fixed any of the vulnerabilities after taking control of the application [4].

Over time, Yahoo moved the authentication scheme for Yahoo Voices over to its own logon system but did not perform many security fixes beyond that. Unfortunately, as was revealed in the attack, they did not remove the unencrypted tables nor did they scan for SQL injections in the application. These two issues, combined with the amount of time the vulnerabilities were exposed to the Internet at large meant that the attackers were able to discover information about the databases used by Yahoo and the data stored within [5].

Once the information was in the wild, Yahoo lost the opportunity to become the primary source of information about the attack. It was unable to control what was reported and what details were released when the attackers themselves and security experts began putting the pieces together and reporting it themselves. The statements released by Yahoo were limited to an acknowledgement of the posting and that there was an investigation and fix action under way [6].

## Real-Time Information Assurance

Modern network defense involves reconfiguring automated defenses to prevent attacks and close holes uncovered in previous attacks. Whenever an attack occurs, trained incident responders comb the logs and network for any details about the attack. They identify the exploit, possibly gather details about the attacker, and refine the network devices to prevent that attack from happening again. Whether it involves updating a blacklist, providing recommendations for an emergency patch or evaluating if an application is so insecure that a portion of it needs to be disabled or hidden to protect the network as a whole, all of these techniques are used reactively.

In order to protect their infrastructure and security processes, Yahoo released only the details that were required by law and did not confirm or deny any additional details released by third parties. Based on the responses and news releases following the attack, we cannot say whether they followed industry best practices when closing security holes identified in this attack, but we can explain those best practices that Yahoo could have followed in response to a compromise of their application.

After an attack, a forensic investigation will be launched with the goals of gathering as much information about the attack to secure the network and application as well as gathering information about the attacker's origin and motives. During the preservation and acquisition portions of the response network hardware, servers, and other items which contain logs or information about the attack will need to be protected from further alteration. This may include disconnecting network systems causing disruptions and outages of services. If all systems are left connected to the network until all the evidence of the attack is preserved, the network operators will need to ensure that the live environment does not overwrite or alter critical information. Identifying and maintaining data sources will require cooperation between the application maintainer, the network support technicians, and any other experts that may be able to provide inputs into locations of potentially relevant data.

From the logs and application itself, the forensic expert or persons responsible will have to compile all potentially relevant data and sift through the log files to reconstruct a sequence of events. There may be hundreds of thousands or millions of log entries captured as part of the data collection process and the analyst will have to identify which entries are normal operations and which entries are potentially relevant. From those relevant entries, the analyst will build a timeline of events in order to determine the extent and execution of the attack. The sequence of events will also point to the weaknesses used by the attacker to gain access to the compromised system and identify any other compromised systems or planted malware.

Once the results of the forensic investigation are complete, the network administrators can begin to patch holes at the network layer by disabling IP addresses at the firewalls and limiting traffic to the impacted system. One new protective measure that has seen widespread press is the Web Application Firewall (WAF). Just as a firewall sets up whitelists and blacklists which allow or deny blocks of IP addresses and allowed ports, a WAF is a device that learns expected HTTP commands and blacklists HTTP commands that may contain malicious logic.

It is possible to configure a WAF by hand with the results of a forensic investigation, and the WAF will forever prevent that same attack from occurring again, but WAFs share the same limitations as blacklists in that unknown vulnerabilities cannot be prevented. To address this, existing vulnerability scan results can be converted into a WAF rule set. A dynamic vulnerability scan can test every field in an application for susceptibility to known attacks such as SQL injection, cross-site scripting, and buffer overflows. When integrated into a WAF, the results can then provide a basic level of protection against known types of attacks.

If a dynamic scan is not run, most WAFs have a learning mode that can identify expected behavior over a period of time. Unfortunately, a deployed application may undergo constant attacks and certain exploits might be allowed in rules created during that learning period. Once the rules are created, they will be monitored and the operator can view any requests that may violate them to determine if a rule will cause issues. All of the rules and the results of their monitoring phase will need to be evaluated.

In order to get the maximum utility out of a WAF, the operators must be knowledgeable in several areas that include the specific application behavior and application security in order to ensure that rules will close security vulnerabilities while maintaining functionality of the application. Just as network firewall rules implemented to improve security can prevent legitimate traffic from passing, WAF rules can disable application functionality by preventing legitimate data from passing. However, sometimes functionality may need to be disabled in the name of security.

Once an attacker has identified a critical security vulnerability that can bring down a system or reveal sensitive information, the assumption is that they and other attackers will use that vulnerability again. Sometimes the hole that was used cannot be fixed without rewriting source code or re-designing the application and the only way to prevent the loss of further data is to disable the vulnerable functionality. This is reserved for cases where a self denial of service is preferable to further compromise.

Unfortunately the users are left to fill the gap in functionality with workarounds or manual processes that can cost an organization more man hours than any investigation or hotfix action. After taking into account the massive costs endured by an organization following a successful attack, the increased development costs, schedule time, and additional coding requirements of a rigorous software assurance process begins to make sense for both software developers and their customers.

In Yahoo's case, the vulnerability identified in their investigation was fixed and the unencrypted username and password database was removed. However, had Yahoo executed a Software Assurance Assessment prior to integration of the Yahoo Voices application into their environment, the injection vulnerability and the insecure database would have been identified before it was exploited. By not proactively eliminating vulnerabilities, Yahoo's customers and public image suffered.

## Software Assurance

Real-time information assurance tools such as a WAF offer program offices a measure of security. They provide very specific protections to an application, but if an attack is not defined by their threat identification systems and a security hole still exists within the source code, that application is still vulnerable. So how does a program office solve this problem? The integration of a Software Assurance policy that includes both dynamic and static analysis of the application source code is essential to preventing exploitation of those vulnerabilities not covered by WAF rules and filters.

There are many ways Software Assurance can be integrated into the development lifecycle of an application, but they all come down to two core implementation methods: the deep dive approach and the triage approach. Both have their merits and drawbacks. It is up to program managers and technical leaders to discuss both approaches and determine which is best for a given application.

## The Deep-Dive Approach

The core idea behind the deep-dive approach is to identify as many security issues as possible through a combination of static and dynamic analysis and manual penetration testing. Every method of every class in every file must be scrutinized and declared vulnerability free. Even the system architecture, risk management procedures, and systems engineering methodologies are evaluated for security vulnerabilities. No attack surface is left unexamined. As a result, this approach is often quite time consuming; taking anywhere from 8 to 12 weeks to complete.

A deep-dive is often conducted toward the end of a lifecycle when active development has finished, but before QA testing begins. Once done, the application can be expected to pass rigorous security testing and is ready for deployment. This approach works best when used with large-platform, MAC I –type systems utilizing more classical development lifecycles such as the Waterfall lifecycle.

Because the costs in both true dollar amounts and man-hours can be extensive, it is necessary to such programs to plan for deep dive approaches in the planning and requirements phases of their development lifecycles. Even with proper planning, the costs and time needed to perform a deep dive security analysis makes this approach prohibitive for most standard applications in use in the DoD today.

## The Triage Approach

In today's fast-paced development environments a Software Assurance approach that takes weeks to complete just is not feasible. Program offices utilizing RAD or Agile development cycles simply cannot allocate more than a week to Software Assurance practices. In such cases, a triage approach may be best.

Time is not the only resource saved by the triage approach to Software Assurance. The material costs are also dramatically lower. Triage is an overall cheaper alternative to the deep dive security analysis of a system. As a result, it lends itself to legacy systems (those systems that have entered into the maintenance and sustainment portions of their lifecycles) where funding for extensive testing, evaluation and repair is often not available.

A triage approach to Software Assurance emphasizes identifying and remediating low-hanging fruit through static and/or dynamic analysis. In this approach, it is not so important to identify and remediate every single security threat, only the easiest or those deemed to be the highest risk should be considered. As each iteration of the lifecycle completes, more and more issues will be identified and remediated.

While this issue takes significantly less time than the deep-dive approach to Software Assurance, there is one glaring flaw: if a high-risk security issue is not properly identified, the application could remain vulnerable through several development iterations. This is where real-time IA measures, like WAFs, can help mitigate the risk to an application.

Consider the Yahoo case study presented earlier: Associated Content was not a large mega-corporation capable of spending millions of dollars to implement a deep-dive Software Assurance program. At the time of their acquisition by Yahoo in May of 2010, the triage approach to Software Assurance was still in its infancy, but it could have saved Yahoo from both the financial and reputation losses it undoubtedly suffered.

A triage of the Associated Content software would have revealed the SQL Injection vulnerability that allowed attackers to retrieve the unencrypted passwords and e-mail address of the systems users. Real-time Information Assurance measures could have been put into place that would have prevented the exploitation while Yahoo developers took a closer, more analytical, look at the vulnerability that would have revealed the lack of proper encryption of the data stored in the legacy database. Unfortunately, many of the developers, managers, and Information Assurance personnel at Yahoo may not have known that a Software Assurance plan was a necessity because they lacked the proper training to identify such risks.

## Education is Key

Regardless of which approach is used to identify, catalog, and remediate security vulnerabilities, it may all be wasted effort without proper education and training of program personnel. While security is becoming a hot topic to teach as part of a software and computer engineering program, the majority of programmers have not yet had much exposure to software security and secure development training. Ideas such as data validation for security and whitelists are foreign concepts to a large percentage of the software development workforce. As a result, it becomes necessary to ensure that everyone involved in the development lifecycle receives a degree of training concerning Software Assurance. It is imperative that developers and testers understand how to identify security risks to the application and how to remediate those risks.

In the end, it is equally important that IA personnel and program management learn the language of Software Assurance so they can communicate openly with the developers and testers about the sort of risks associated with different vulnerabilities. This can only be achieved through repeated exposure to education and training materials, and an open dialog about the overall security profile of an application.

Combining Software Assurance and Real-Time Information Assurance measures helps to ensure that security gaps in an application's source code are covered, at least temporarily, by the rules governing the WAF. But, once a WAF is in place, why bother repairing those gaps? As mentioned earlier, WAFs and other Real-Time Information Assurance measures essentially amount to security blacklists: defining what harmful information should look like and blocking it at the server and/or application layers. As attackers refine their techniques and new, more creative methods to defeat these blacklists are created, the rules governing what tainted or harmful data looks like will need to change.

If security flaws are not fixed at the source, engineers will find themselves in an arms race trying to keep Real-Time Information Assurance rules updated to defend against current attack vectors. Repairing the security holes at the source, when done correctly using whitelists and proper data validation, puts an end to the arms race by eliminating the security threat. But may all be for naught if the developers, testers, program management, and Information Assurance staff are not properly educated regarding Software Assurance and the risk that not integrating such principles into the development lifecycle can bring.

Software Assurance is not just an option to secure our critical software applications, it is an absolute necessity. The DoD must be proactive when it comes to identifying and assessing the weaknesses present in software in use by the warfighter and our national infrastructure. Building Security in to the software lifecycle is critical to this task, especially if it is built in early and often. Doing so will prevent cyber attacks to critical systems and save millions of dollars each year. Not just a solution for new software being developed, triage approaches to Software Assurance allow legacy systems to incorporate these techniques into their sustainment plans improving the preparedness of the nation as a whole to all cyber threats.

**Disclaimer:** The views expressed in this article are those of the authors and do not necessarily reflect the official policy or position of the US Air Force, Department of Defense or the U.S. Government.

## ABOUT THE AUTHORS

Robert A. Voitle, Jr. has been a contractor working as a Senior Software Assurance Analyst for the Air Force at the Application Software Assurance Center of Excellence since 2008, and is regarded as an expert on Software Assurance and application security. He has worked as a consultant for Fortify Software, Aspect Security and, currently, Cigital Inc. He has authored several works that range in scope from informal news articles to white papers. Robert is a graduate of Auburn University where he studied Journalism and Technical and Professional Communication.

**E-mail: rvoitle@cigital.com**

First Lieutenant Arthur J. Boote is currently assigned to the Business & Enterprise Services PEO, Gunter Annex in Montgomery, Alabama serving as the Chief Technology Officer for the Application Software Assurance Center of Excellence. 1Lt Boote has previous experience as a Cyberspace Crew Commander for the 561 NOS and Digital Forensics technician prior to joining the Air Force. 1Lt Boote was instrumental in the publication of the Secure Programming Best Practices Guide for Department of Defense programmers. 1Lt Boote has a degree in Computer Engineering from Florida State University.

**E-mail: arthur.boote@gunter.af.mil**

James "Woody" Woodworth is a Department of the Air Force Civilian currently assigned to the Business & Enterprise Services PEO, Gunter Annex in Montgomery, Alabama serving as the Chief of the Application Software Assurance Center of Excellence (ASACoE). He has served in the Air Force for 33 years, the first 20 years on active duty in the Air Force JAG Corps and the remaining 13 as a civilian in a variety of software development positions. He has been assigned to the ASACoE since its stand-up in 2007, first as the Operations Chief and then as Chief upon his return from deployment to Iraq. He is considered one of the leading Software Assurance Subject Matter Experts in the Department of Defense.

**E-mail: james.woodworth@gunter.af.mil**

## REFERENCES

1. National Defense Authorization Act (NDAA) for Fiscal Year 2013. Pub. L. 112-239 §933. 2 Jan 2013.
2. Schwartz, Mathew. "Yahoo Hack Leaks 453,000 Voice Passwords." InformationWeek. UBM Tech, 12 Jul 2012. Web. 29 May 2013. <http://www.informationweek.com/security/attacks/yahoo-hack-leaks-453000-voice-passwords/240003587>.
3. Heist, Matt. "What Yahoo needs to do with Associated Content." The Digital Content Blog. Guardian News and Media Limited, 08 Jun 2010. Web. 29 May 2013. <http://www.guardian.co.uk/media/pda/2010/jun/08/yahoo-associated-content>.
4. Arthur, Charles. "Yahoo Voice hack leaks 450,000 passwords." thegaurdian. Guardian News and Media Limited, 12 Jul 2012. Web. 29 May 2013. <http://www.guardian.co.uk/technology/2012/jul/12/yahoo-voice-hack-attack-passwords-stolen>.
5. "Yahoo! Voices Website Breached 400,000 Compromised." TrustedSec. TrustedSec, 11 Jul 2012. Web. 29 May 2013. <https://www.trustedsec.com/july-2012/yahoo-voice-website-breached-400000-compromised/>.
6. Prince, Brian. "Yahoo Confirms 400,000 Passwords Stolen in Hack." eWeek. Quinstreet Enterprise, 07 Jul 2012. Web. 29 May 2013. <http://www.eweek.com/c/a/Security/Yahoo-Confirms-400000-Passwords-Stolen-in-Hack-112338/>.

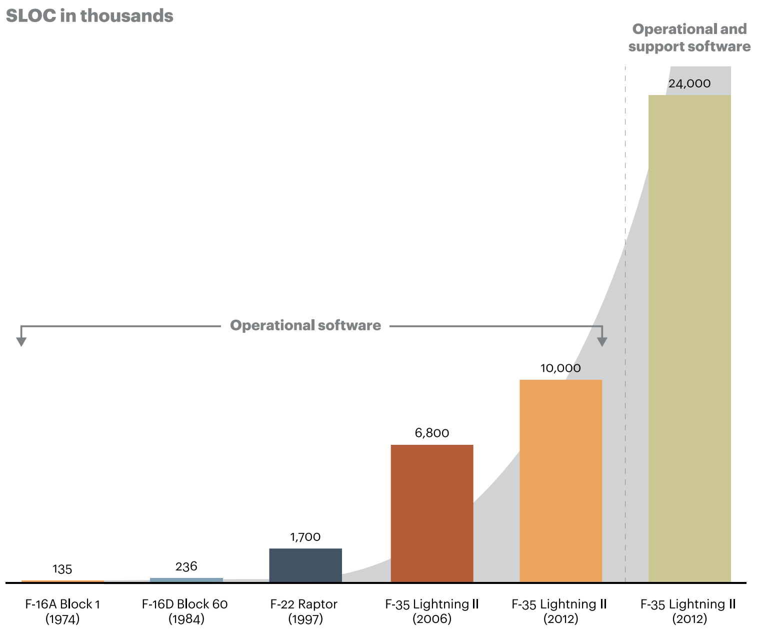
# Effective Approaches for Delivering Affordable Military Software

**Christian Hagen, A.T. Kearney**
**Steven Hurt, A.T. Kearney**
**Jeff Sorenson, A.T. Kearney**

**Abstract.** As the U.S. military shifts its focus from metal and mechanics to developing and integrating control systems—unmanned vehicles, drones, and smart bombs—delivering advanced software-enabled systems affordably will become a vital success factor in weaponry.

Technology is a strategic weapon indispensable to the viability of virtually all industries and organizations. Nowhere is this truer than in the military and the DoD. The DoD is well known for its hardware-based weaponry, which is unmistakable in aircraft, satellites, missiles, and other systems in its arsenal for defending the country. Now, the military is shifting its emphasis from hardware to software. More specifically, the DoD wants to better manage the development of software that plays a growing role in national security, while improving the affordability and quality of this increasingly crucial element of modern weapon systems.

This shift will require the DoD to alter its perspective, processes, and capabilities to avoid the increasing costs associated with software development, modernization, and sustainment. Failure to make these changes will burden the country's capabilities and force the military to operate under complex and tangled systems that will cost billions of dollars to revamp. As one U.S. Air Force general explains it, "The B-52 was judged by the quality of its sheet metal; the F-35 will be judged by the quality of its software."



Notes: SLOC for F-16 and F-22 are at first operational flight. F-35 SLOC figures are from first test flight and current estimates/sources.

Sources: P. Judas and L.E. Prokop, "A historical compilation of software metrics with applicability to NASA's Orion spacecraft flight software sizing," Innovations in Systems and Software Engineering, vol. 7 issue 3, September 2011. p. 161–170; Andrea Shalal-Esa, "Pentagon focused on resolving F-35 software issues," Reuters, March 2012; Robert N. Charette, "F-35 Program Continues to Struggle with Software," IEEE Spectrum, September 2012

*Figure 1: The number of SLOC has exploded in avionics software.*

As advanced software systems and embedded software technologies become the brains behind modern warfare, our military is paying to develop and maintain exponentially larger systems. When measured by source lines of code (SLOC) created or modified by software developers, the amount of software code in modern war-fighting systems has increased significantly over the past decade (see figure 1).[1]

Like it or not, the DoD is now in the software business.

## The Business of Software

The increased demand for software stems from an emphasis on advanced operational capabilities and requirements. For example, the airframe for advanced fighter jets has reached the limit of modern aerodynamic improvement. This limitation, coupled with recent advances in remotely piloted aircraft, places greater demands on Operational Flight Programs (OFPs) to rapidly bring new capabilities to bear. Similarly, smart weapons—such as the GPS-guided Joint Direct Attack Munition, laser-guided munitions, and AIM-9X—require continual software updates to stay smart. Full autopilot, launch accessibility region calculations, and other advanced capabilities can only be enabled using increasingly sophisticated algorithms and software architectures.

Additionally, ground systems need to overcome the challenges associated with deploying large-scale software solutions. The Army recently canceled the Future Combat Systems (FCS) program due to budget and schedule issues that were largely the result of complex software development problems. Recent reviews of the F-35 Joint Strike Fighter underscore the challenges of FCS and many other military programs, illustrating the difficulties program offices face in managing, adapting, and even comprehending software development initiatives.[2,3]

In June 2012, the Air Force Scientific Advisory Board released a report that highlighted the growing role of software in sustaining aircraft over the long term and noted that software's utility and complexity has grown faster than the Air Force's ability to address it across a system's lifecycle. So while hardware sustainment costs will decrease as the USAF reduces the number of aircraft, software sustainment costs will not follow suit because a weapon system generally needs the same software sustainment whether it is a fleet of 10 aircraft or 1,000 aircraft.[4]

With shrinking defense budgets and increasingly complex system requirements, government leaders and key contractors are struggling to deliver advanced software-enabled systems affordably. The shift from hardware to software is particularly challenging for modernizing systems and managing sustainment.

## Defense System Software: Looking Ahead

The defense and software landscape is dynamic, with several technological, programmatic, and enterprise barriers, among other issues, having a major impact over the next decade (see figure 2).

As software continues to become more important to the military it will be imperative to drive savings and efficiencies into both new development and maintenance. Four areas of advantage—architecture, commercial software, should-cost analysis, and organic sustainment—are indispensable to realizing this savings without impacting the required performance and have the following objectives:

**The architecture advantage.** Determine the optimal architecture decisions and impact early in the design and planning processes, and continue assessments through the maintenance phase.

- Recognize that selecting the proper software architecture is a cost-effective way to rapidly improve capabilities.
- Understand the impact that modular, federated, and integrated architectures have on weapon systems and future capabilities.
- Determine which architectures allow for cost-effective enhancements.

**The commercial software advantage.** Upgrade capabilities, accelerate deployment, and improve total cost of ownership (in just the same way as the private sector does).

- Evaluate how and where commercial software can be deployed to use the latest capability at the lowest cost.
- Synchronize, update, or utilize refresh cycles to take advantage of the latest commercial software functionality.
- Use commercial software in both critical and non-critical systems as a replacement for costly custom development.

**The should-cost analysis advantage.** Perform should-cost analyses to estimate costs more accurately for the software capabilities being produced, integrated, and tested—knowing this can save millions of dollars on key projects.

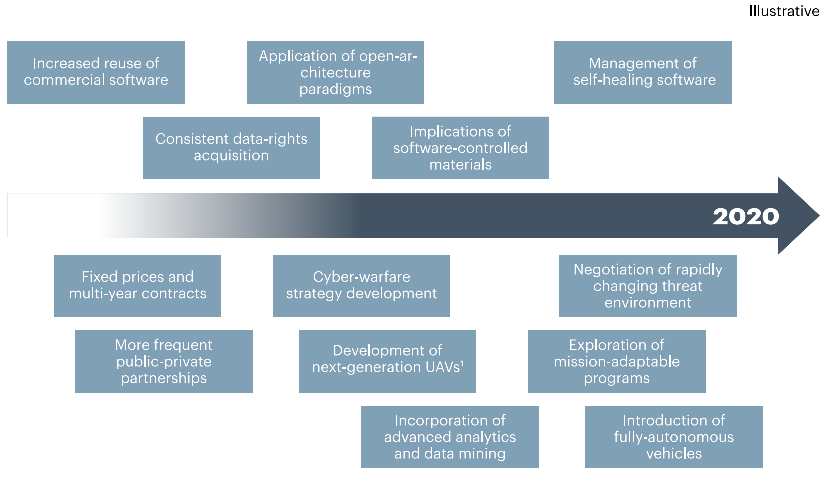
- Validate contractor proposals by independently estimating the cost of software development.
- Identify improvement opportunities for software developers to reduce costs and deliver benefits to the program and taxpayers.
- Ascertain the software models and comparisons needed to get the most accurate cost estimate.

**The organic sustainment advantage.** Outline a clear strategy for the development of organic capabilities needed to maintain the weapon systems—and do so affordably.

- Add new capabilities to existing weapon systems to maximize asset value.
- Determine the right partnership between industrial and organic sustainment efforts.
- Build an organic capability to support next-generation weapon systems.
- Optimize investment in software maintenance and modernization.

## The Architecture Advantage

How various software-enabled functions are designed and constructed within a weapon system determines the degree of effort required to enhance or modify software and deliver new capabilities. Some integrated architectures are so tightly coupled that making even small changes incurs significant costs. However, if modularity and loosely independent design are applied from the start, maintenance and enhancement efforts will be more efficient.[5] One approach to affordability promotes the use of federated avionics architectures, such as the one exhibited on recent fifth-generation fighter jets.


Illustrative
Increased reuse of commercial software
Application of open-architecture paradigms
Management of self-healing software
Consistent data-rights acquisition
Implications of software-controlled materials
2020
Fixed prices and multi-year contracts
Cyber-warfare strategy development
Negotiation of rapidly changing threat environment
More frequent public-private partnerships
Development of next-generation UAVs¹
Exploration of mission-adaptable programs
Incorporation of advanced analytics and data mining
Introduction of fully-autonomous vehicles

¹Unmanned aerial vehicles
Source: A.T. Kearney analysis

*Figure 2: Issues likely to impact defense software between now and 2020.*

**Federated architecture.** Federated architectures allow information to be shared among modules. The design is component-based and similar to that of the old legacy systems. Federated systems typically have a Central Processing Function (CPF) that coordinates information across the platform. In avionics software, the CPFs do not assume responsibility for data processing but instead organize information necessary for pilots to make decisions, while each subsystem gathers and processes data from its relevant sensors. A federated architecture delivers the same advanced capabilities and sensor fusion of highly integrated architectures but with somewhat reduced overall processing efficiency because of its modularity. However, today's modern hardware has ample capability and speed to overcome any reduction in the processing efficiency of federated architectures.

Despite decreased efficiency, modular components can be upgraded quickly and independently, thereby reducing maintenance costs. Because of the nature of designs featuring tight coupling, developing and maintaining integrated architectures across the software development lifecycle requires significantly more effort. By comparison, the overall required effort and associated costs of software designs for federated architecture are lower by a factor of two or more (see figure 3).

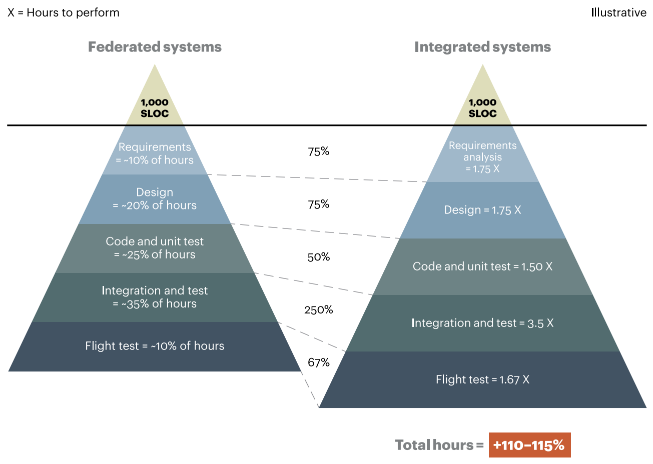


*Figure 3: Integrated systems require more effort to maintain than federated systems.*

Note: SLOC refers to source lines of code. 110–115% is the increase in the number of hours required to code 1,000 SLOC in an integrated system versus a federated system—found by taking the weighted average of complexity factors across each phase of the software development life cycle

Source: A.T. Kearney analysis

**Open architecture.** The next significant evolution in avionics architecture will rely on the principles of open architecture, which offer an alternative to today's multitude of proprietary standards. Open architecture is a type of hardware or software architecture that allows components conforming to agreed-upon standards to be added, upgraded, and swapped. Open architecture allows independent parties to design and develop interoperable components that work together under the specified standards. By applying similar functions or skills at the platform level, greater specialization and flexibility becomes possible among developers across multiple contractors—significantly reducing the cost and effort of any software project.

**Open federated architecture.** A federated architecture is the natural model for open architecture development. The combination of the two paradigms (open and federated) increases the ability of development teams—and even encourages them—to modularize their code and compartmentalize their efforts.

Already, major contractors are developing their own native open architecture standards and models focused on subsystem development. However, the government should promote the development of a general open architecture, supported by all involved contractors across entire systems. Such a coordinated effort would encourage general adoption. The DoD should focus early design on open architectures while striving to achieve needed performance capabilities for the weapon systems. Great benefits can be realized by investing time early in the architecture design phase with contractor and government software professionals who will maintain the system and live in the future with today's choices.

## The Commercial Software Advantage

Just as private sector companies have transitioned to commercial software and enterprise resource planning systems over the past two decades, the military is now undertaking a similar transition. When appropriate, using software developed for commercial use instead of expensive, time-consuming, custom-developed software can bring the latest capabilities to the battlefield and support systems at a fraction of the cost. Using commercial software in noncritical applications allows proven best practices to be adopted, thus avoiding problems already solved commercially.

In recent years, commercial software has become more attractive for modifying or retrofitting existing platforms for future weapon systems. Commercial software applications and processes draw from a larger pool of experience to speed up development at a lower cost. Additionally, commercial platforms offer advanced capability and increased performance over custom or embedded systems, which often rely on older technology.

As an added benefit, the extensive use of commercial software presents an opportunity to coordinate upgrades across weapons platforms by sharing common architectures, code, and maintenance efforts. Commonality across systems will allow for more centralized training of both in-house maintainers and operators, which will increase the flexibility of personnel across the services, essential in times of conflict or force reductions. Additionally, larger purchases of commercial software will offer significant negotiating power because of the supplier's near-zero marginal cost of producing or delivering additional units.

Transitioning from custom to commercial software begins by understanding their basic differences. Commercial software has four key advantages:

**Faster development cycle.** Commercial software is developed much faster than traditional DoD software systems, with comparable system developers being 50 to 100 percent more productive.[6] The shorter cycle allows advanced capabilities to be fielded faster and development teams to respond swiftly to requests for changes.

**Advanced capabilities.** Modern software development methods are changing rapidly, providing developers with a wider range of tools for coding applications. This allows for the use of the latest techniques to deliver applications that would be impossible to develop with legacy software systems.

**Lower cost.** Only a few software developers have proficiency with DoD custom development standards. As a result, it is expensive to retain a large number of highly experienced developers for even basic projects. Recruiting from the much larger pool of developers with commercial software experience will drive down costs to market rates and yield a more agile pool of developers.

**Continual refresh.** Commercial software undergoes continual maintenance, which leads to new capabilities with each software release. These upgrades are aligned with functional best practices and offer customers the benefits from global leaders that create leading and scalable functionality for end users and organizations.

Capitalizing on the benefits of commercial software requires adopting a faster, more coordinated refresh strategy. A hardware refresh that is closer to the three- to five-year cycle for commercial platforms should replace the traditionally longer defense modernization schedules, which take 10 or more years for many DoD projects. A shorter cycle ensures that large investments in war-fighting systems remain operational throughout their lifetime.

Executing this more rapid refresh cycle will require secure sources of regular funding to reduce the total cost of ownership. If funding is not provided for a full upgrade and the modification cycle lags, there will be additional costs to maintain legacy configurations. Only a stringent schedule of regular updates will manage risk and reduce overall costs.

To summarize, the use and effective integration of commercial software is not only attractive but entirely feasible for DoD and weapon systems. Improving commercial hardware and software solutions can greatly enhance capabilities, increase the speed to deployment, and significantly drive down costs. The DoD's ability to recognize this shift and manage the vendors and solutions will be crucial over the next decade and beyond.

## The Should-cost Analysis Advantage

For most software development projects, estimating the required cost and effort can be challenging. Procurement and acquisition organizations—long the experts at estimating hardware-related projects, upgrades, and modernization efforts—often struggle to estimate and manage software development costs. Several authors have developed approaches and estimates for software engineering effort, project duration, and quality.[7,8,9,10,11,12,13,14,15,16,17,18] Not surprisingly, given the complexity of software development efforts, others have offered critiques of these approaches.[19,20,21,22,23,24,25]

An inaccurate estimate can cost time, money, and lost capabilities. It is difficult to estimate software development efforts largely because of the complexity of the domain coupled with historically inadequate sizing estimates, a lack of experience with similar efforts, and poorly defined requirements. Poor estimation of software development effort and complexity is a cause of DoD program cost growth and schedule overruns.[26]

Commercial projects consistently go over budget and underdeliver on functionality. Add in defense and avionics complexities, and gauging total software costs becomes quite complex. By applying standard techniques of software effort and cost modeling, an organization can accurately estimate the required work and associated cost to avoid software development overruns. This has recently been emphasized by many senior leaders across each of the services and broader DoD.

A software should-cost review is an opportunity for everyone involved to question the traditional ways of doing business and improve efficiency across the value chain. Furthermore, when targeted to a specific program's needs, should-cost is a valuable tool for reducing costs without eroding supplier profits or cutting capabilities. It is a win-win proposition for both the DoD and its suppliers, providing a means to realize the "do-more-without-more" philosophy championed in recent years by the Office of the Secretary of Defense.[27] The analysis can improve transparency and affordability by breaking the cycle of history-based cost estimation. In doing so, it answers the overriding question: What would the costs of a program be in an efficient, highly competitive environment?

A should-cost analysis provides insight into cost drivers (for use in planning and negotiations) and forces accountability, especially across contracts and suppliers—shedding light on the overall development lifecycle. Typical should-cost software estimation includes the following four elements:

**Parametric modeling.** Models based on SEER for Software (SEER-SEM) and Constructive Cost Model, which rely on industry standards drawn from experience across multiple projects, help estimate the software effort and determine optimal schedules.

**Bottom-up estimation.** Each activity is modeled based on a discrete view of component costs against specific industry-standard benchmarks and cost drivers.

**Contractor comparison.** Industry benchmarks are a good way to estimate what software should cost. Examining the performance of similar contractors helps develop an understanding of best-practice performance.

**Program comparison.** Benchmarking similar services and common mission profiles across a comparable scope of work can help estimate costs.

Bottom-up modeling of avionics software development starts with the basic building blocks, including people (teams), processes (software development life-cycle phases), and technology (software packages), broken down into the smallest components to gain insight into the system as a whole.

Once the system has been defined at its lowest possible level—typically down to individual programmer teams or subsystem components—cost drivers are applied against specific measurements and activities. A typical measurement might be hours required to develop equivalent source lines of code (ESLOC) for a given team. The cost drivers model the effort required for a team to develop a particular piece of software. Summing these cost estimates over each team, phase, and software package provides a full view of the effort required across people, processes, and technology.

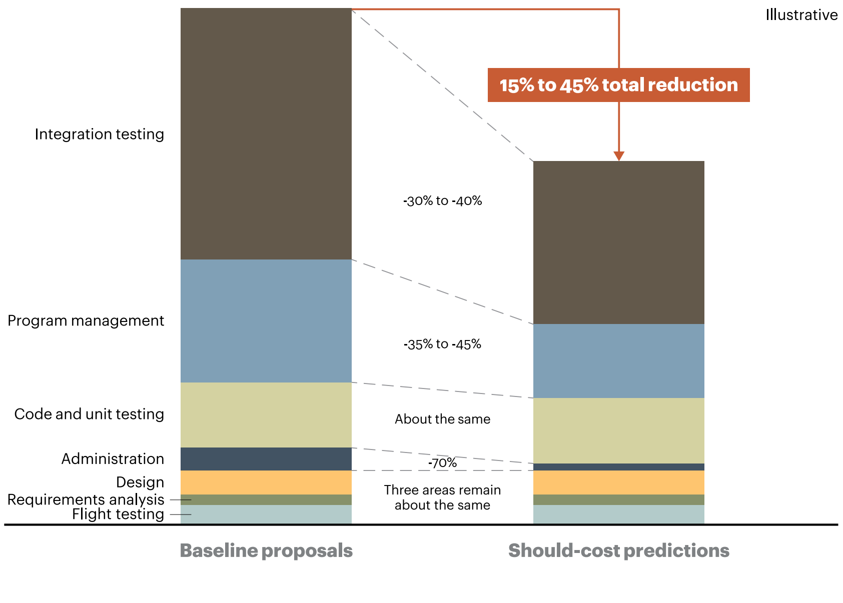
CIVILIAN TALENT IS MISSION-CRITICAL.
LET'S GET TO WORK.
Work for Naval Air Systems Command (NAVAIR) and you'll support our Sailors and Marines by delivering the technologies they need to complete their mission and return home safely. NAVAIR procures, develops, tests and supports Naval aircraft, weapons, and related systems. It's a brain trust comprised of scientists, engineers and business professionals working on the cutting edge of technology.
You don't have to join the military to protect our nation. Become a vital part of NAVAIR, and you'll have a career with endless opportunities. As a civilian employee you'll enjoy more freedom than you thought possible.
Discover more about NAVAIR. Go to www.navair.navy.mil.
Equal Opportunity Employer | U.S. Citizenship Required
NAVAIR CIVILIAN
CHOICE IS YOURS.

## Should-cost: A Detailed Action Plan

A should-cost analysis alone will not reduce costs; rather, the savings are achieved by incorporating the results and implementing key initiatives. A successful software should-cost analysis includes a detailed action plan with specific milestones for reevaluation and an aggressive implementation mindset to prevent the should-cost from being merely a theoretical study (see sidebar: Case Study: U.S. Air Force Avionics Should-cost).

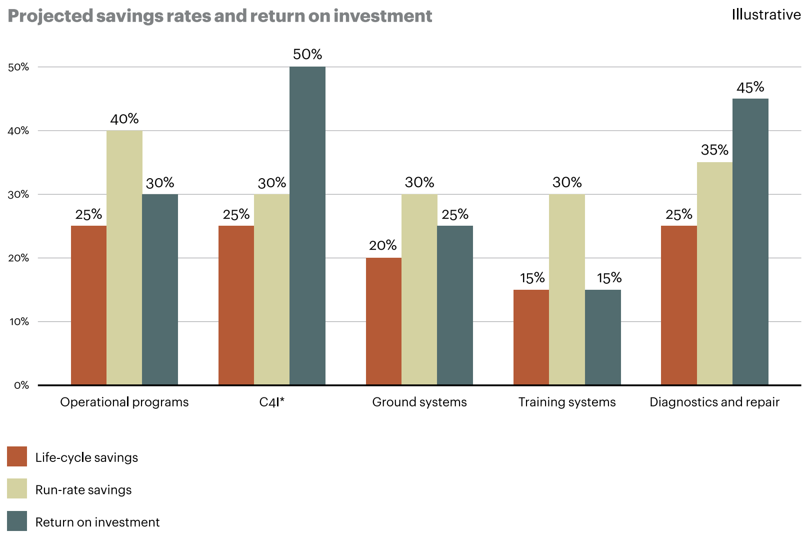
Following are five ways to ensure that a should-cost analysis delivers accelerated, maximum benefits:

**Bring best practices to bear.** Start the should-cost analysis with an aggressive attitude for challenging the status quo. Look outside the current paradigm for best practices to replicate in a competitive environment. Focus on determining the most efficient cost-to-deliver program requirements—not on the likelihood that these requirements will be accepted. Continually ask "what if?" and "why not?"



Source: A.T. Kearney analysis

*Figure 4: Should-cost modeling identifies significant potential savings.*



| | Life-cycle savings | Run-rate savings | Return on investment |
|---|---|---|---|
| Operational programs | 25% | 40% | 30% |
| C4I* | 25% | 30% | 50% |
| Ground systems | 20% | 30% | 25% |
| Training systems | 15% | 30% | 15% |
| Diagnostics and repair | 25% | 35% | 45% |

*C4I refers to command, control, communications, computers, and intelligence
Source: A.T. Kearney analysis

*Figure 5: Organic sustainment offers considerable savings opportunities.*

**Perform rigorous analysis.** Acquire an in-depth understanding of the root-cost drivers and efficiency potential for major areas, including supply chain, manufacturing, program management, and overhead. Detail the savings potential to support the conclusion and drive tangible action.

**Establish the right incentives.** Recognize that the proper incentives, benefiting both the government and its suppliers, will encourage people to move beyond the status quo and act with appropriate urgency. Use a collaborative effort to bring the best results. Set up incentives to encourage everyone to perform in a way that improves suppliers' profits while lowering government costs.

**Translate opportunities into tangible action.** Convert cost-management opportunities into realistic action plans with clear time lines and responsibilities. Use multiple approaches to reduce costs, including negotiation, investment, joint-process improvement, and contract restructuring.

**Track performance against the cost-reduction plans.** Implement a target assurance program to identify cost-reduction targets and milestones. Review progress regularly to understand performance slips and ensure that mitigation steps are in place. Make sure progress is transparent, credible, and well managed.

A should-cost method offers a tangible value proposition for avionics software development. In fact, 15 to 45 percent of the cost of avionics development can be removed from the system by shedding light on the development process, isolating opportunities at the subcomponent level, and increasing the fact base for negotiation (see figure 4). The cost savings are long-term because the improvements carry over multiple development cycles and modernization efforts within a program.[28]

## The Organic Sustainment Advantage

Software maintenance is a growing portion of the post-development work needed to enhance and sustain weapons platforms. As software becomes more common in acquisitions, contractors' past sustainment efforts must give way to more cost-effective and efficient government-led sustainment. With multiple service-life extension programs in effect for legacy platforms, increased government software sustainment will free contractors to focus on modernizing to keep pace with the rapid advances in sensor and weapons technology. In most instances, the government can maintain a stable sustainment organization at a considerably lower cost than primary contractors.

**In-house maintenance.** The government mandates that a significant portion of weapon systems sustainment be conducted in-house to preserve a captive capability that can be called upon in times of war.[29] This recent legislatively enforced transition to organic sustainment makes sense and offers significant savings opportunities—for instance, the government can often provide lower wage and overhead costs in addition to payback periods of fewer than five years. For example, the three Software Maintenance Groups (SMXGs) across the Air Force provide services from engine software design[30] to full OFP block-cycle development on platforms that range from the F-16 to MILSTAR satellite control stations.[31] Estimates for sustainment initiatives range from 15 to 30 percent in lifecycle savings over the next two decades when compared to full contractor sustainment. Additionally, annual run-rate labor savings are estimated between 30 and 40 percent following transition for representative programs (see figure 5).

Finally, through increased use of public-private partnerships, programs can anticipate further benefits to core hour requirements and capability targets while reducing risk through extended contractor support.

Organic resources are typically capable of conducting much more sophisticated sustainment activities than is often presumed. The DoD should undertake capability studies to evaluate the potential of the existing organic resources and organizational readiness to accept future workloads. Modernization and software upgrade schedules should be aligned and the result embedded into a capability road map to govern overall schedules and prioritization.

**Data rights acquisition.** To enable in-house software maintenance, appropriate government rights to the data are critical. Funding for data rights acquisition should be earmarked as an essential expenditure. It may be necessary to fund acquisition independently to prevent potential mingling or re-appropriation that favors imminent challenges over long-term needs. Furthermore, in-house training on the use and relevance of software data rights across program offices and the DoD will ensure that all data and data formats acquired are usable throughout the system's life.

**Necessary skill sets.** Several key skills will be needed over the next decade. Advanced work in OFPs, command and control systems (C4I), and advanced ground control stations will augment and replace workloads currently in test stand and control, and verification and validation. Software architecture and engineering skills will be required earlier in the software development lifecycle. Software sustainment will inevitably include ongoing enhancement, which requires early participation in the design process. New skills in requirements analysis, functional design, and software architecture will be required to field these new capabilities. Internally developing and supporting this work will be vital to the ongoing success of organic software sustainment organizations.

By developing a targeted transformation road map to grow the organic sustainment organization and capabilities, the DoD can meet key software maintenance goals that will drive next-generation weapon systems. The organic growth road map should prioritize the transition of avionics and C4I programs to preserve knowledge of the most important and fastest-changing weapon system software. Transitioning programs may result in a 30 to 40 percent run-rate savings following a transition to an 80 percent organic capability.

## Transformation Is Key to Winning the Race

A radical reshaping of software policies and practices will help the U.S. military avoid falling behind in this arena. From remotely piloted aircraft and smart bombs to autonomous vehicles and advanced fighter jets, software is crucial to the success of today's weapon systems. Focusing solely on developing and maintaining military hardware is no longer an option. With shrinking defense budgets and increasingly complex systems, the defense industry and services must fight to deliver modern software-powered weapons affordably. To deliver on this ambitious objective, the military must drastically transform its approach to software. New organizational structures, operating models, and tools will be essential to modernizing and sustaining the U.S. weapon systems. The mission is to deliver benefits today while keeping pace in the race to the future—or risk losing it.

## Case Study

### U.S. Air Force Avionics Should-cost

Aerospace engineers and designers have pushed the bounds of the possible with the mechanical design and construction of military aircraft, creating machines that are both breathtakingly beautiful and devastatingly effective. Now, software developers are pushing the limits of traditional avionics in modern systems to keep older aircraft competitive and create innovative capabilities in the new ones.

On the F-22 increment 3.2A program, according to a recent DoD Better Buying Power fact sheet, the Air Force successfully identified and implemented cost-saving initiatives of 15 percent (equaling $32 million) to address areas in the software development process that were above industry benchmarks.

### Approach

The analysis involved the processes and costs of a proposed half-billion-dollar modernization effort for the Air Force fleet. This software-only modernization effort involved adding several new tactical capabilities to ensure the success of current and future missions.

The project began with an extensive evaluation. In interviews with the primary contractors responsible for developing the modernization software, the team analyzed the development processes, lifecycle, estimation techniques, system components, software development roles, and unique system components involved in an enhancement effort.

From the data collected, the team created a component-based model of the system, down to the smallest development effort. The software effort and cost was evaluated based on the components, development process used, historical productivity, and technology platforms. The result was a complete should-cost model of the entire development effort—from the top down to the subsystem components in terms of people, processes, and technology.

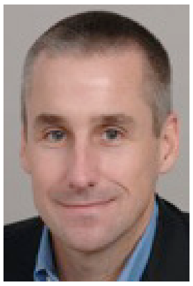
The final model estimated the timing (in hours) for each phase of the software development lifecycle. It created transparency into the development process, targeted DoD areas for improvement, and streamlined cost negotiations for the modernization effort.

### Results

The should-cost analysis identified a 15 to 30 percent gap between the current software development process and a targeted should-cost approach. The gap was further substantiated through a parametric model using standard software industry tools such as SEER and COCOMO.

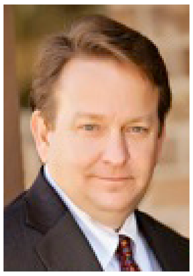
From interviews and data audits, the team identified improvement areas, such as contractor handoffs and inefficient testing during the integration phase, and created action items to reduce hours and costs, achieving the goals set out by the should-cost analysis. A collaborative effort between the military and its contractors successfully improved capabilities at a reduced price—providing value to all parties, including taxpayers.

## ABOUT THE AUTHORS



Christian Hagen is a Partner in A.T. Kearney's Strategic Information Technology Practice and is based in Chicago. He advises many of the world's largest organizations across multiple industries, including government and defense contractors. He specializes in helping clients leverage software and information technology to increase efficiencies and gain competitive advantage. Christian has led several global studies for A.T. Kearney and authored nearly 50 published papers on low-cost competition, software engineering, e-commerce, technology innovation, and strategy.

**E-mail: christian.hagen@atkearney.com**



Steven Hurt is a Partner in A.T. Kearney's Public Sector and Defense Services. Steve has worked with several of the USAF's highest visibility programs to drive affordability in both software and hardware sustainment. Specifically, Steve has focused on should-cost analyses, business case analyses, contract negotiations, and developing business intelligence aimed at reducing cost.

**E-mail: steven.hurt@atkearney.com**

Jeff Sorenson is a Partner in A.T. Kearney's Public Sector and Aerospace Defense Practice based in Washington, D.C. Jeff retired as a Lieutenant General with more than 20 years of acquisition experience. He successfully developed, procured, and delivered over 30 different military systems including battlefield intelligence automation, night vision, and tactical missile systems. As the Army's Chief Information Officer (CIO)/G6, he transformed Army network information technology capabilities to enhance business and war-fighting command, control, and communication systems.

**E-mail: jeff.sorenson@atkearney.com**

## REFERENCES

1. SLOC is used here as the best available metric across DoD programs for estimating the required software development and maintenance effort.
2. United States Government Accountability Office, "Joint Strike Fighter: Restructuring Places Program on Firmer Footing, but Progress Still Lags." GAO-11-325, April 2011; p. 29-31, <http://www.gao.gov/new.items/d11325.pdf>.
3. United States Government Accountability Office, "Joint Strike Fighter: DOD Actions Needed to Further Enhance Restructuring and Address Affordability Risks." GAO-12-437, June 2012; p. 18-20, <http://gao.gov/assets/600/591608.pdf>.
4. Starosta, G. "AFSAB Urges Strategic Shift In Air Force's Long-Term Sustainment Plans." InsideDefense.com, June 2012; <http://insidedefense.com/Inside-the-Air-Force/Inside-the-Air-Force-06/15/2012/afsab-urges-strategic-shift-in-air-forces-long-term-sustainment-plans/menu-id-82.html>.
5. Modularity is the degree to which a system is made up of independent units that can be combined into a larger application.
6. Jones, C. Applied Software Measurement, McGraw-Hill, 2008, Table 3-1, p. 189.
7. Stark, G. (2011). "A Comparison of Parametric Software Estimation Models Using Real Project Data." CrossTalk – The Journal of Defense Software Engineering. January 2011.
8. Boehm, B. (1981). Software Engineering Econo mics. Englewood Cliffs, N.J., Prentice Hall.
9. Boehm, B. (2006). "Minimizing Future Guesswork in Estimating," IBM Conference on Estimation, Atlanta, GA, Feb. 2006.
10. Jones, C. (2007). "Software Estimating Rules-of-Thumb," <http://www.compaid.com/caiinternet/ezine/capers-rules.pdf>, Mar. 2007.
11. Jones, C. (1997). Applied Software Measurement, 2nd Ed., McGraw-Hill, NY.
12. Rone, K. et al. (1994). "The Matrix Method of Software Project Estimation", proceedings of the Dual-Use Space Technology Conference, NASA Johnson Space Center, Houston, TX, Feb.
13. J.W. Bailey and V.R. Basili. "A Meta-Model for Software Development and Resource Expenditures," Proceedings of the 5th International Conference on Software Engineering. New York: Institute of Electrical and Electronic Engineers, 1983.
14. ISBSG, International Software Benchmarking Standards Group, <http://www.compaid.com/caiInternet/ezine/ISBSGestimation.pdf>.
15. ISBSG, International Software Benchmarking Standards Group, <http://www.isbsg.org/isbsg.nsf/weben/Project%20Duration>.
16. McConnell, S. (2006). Software Estimation: Demystifying the Black Art, Redmond, WA, Microsoft Press.
17. International Function Point User's Group (IFPUG), Function Point Counting Manual, Release 3.1, 1990.
18. Jones, C., "Achieving Excellence in Software Engineering," presentation to IBM Software Engineering Group, March, 2006.
19. P. Oman, "Automated Software Quality Models in Industry," Proceedings of the Eighth Annual Oregon Workshop on Software Metrics (May 11-13, Coeur d'Alene, ID), 1997.
20. G. Atkinson, J. Hagemeister, P. Oman & A. Baburaj, "Directing Software Development Projects with Product Measures," Proceedings of the Fifth International Software Metrics Symposium (Nov. 20-21, Bethesda, MD), IEEE CS Press, Los Alamitos, CA, 1998, pp. 193-204.
21. T. Pearse, T. Freeman, & P. Oman, "Using Metrics to Manage the End-Game of a Software Project," Proceedings of the Sixth International Software Metrics Symposium (Nov. 4-6, Boca Raton, FL), IEEE CS Press, Los Alamitos, CA, 1999, pp. 207-215.
22. Kemerer, C. F. (1987), "An empirical validation of software cost estimation models," Communications of the ACM, Vol 30, No 5, pp. 416-429.
23. Jorgensen, M., and Sheppard, M. (2007), "A Systematic Review of Software Development Cost Estimation Studies," IEEE Transactions on Software Engineering, Vol 33, No 1, Jan. pp 33-53.
24. Fenton N. E., and Pfleeger, S. L. (1997), Software Metrics: A Rigorous & Practical Approach, 2nd Ed., London, PWS Publishing.
25. Jorgensen, M. (2004), "A Review of Studies on Expert Estimation of Software Development Effort," Journal of Systems & Software, Vol 70, No. 1-2, pp. 37-60.
26. Lucero, Scott. "Software Sustainment Challenges in Defense Acquisition." Office of the Deputy Undersecretary of Defense: Acquisition and Technology. April 2009. p. 3. Accessed 30 May 2012. <http://www.acq.osd.mil/se/webinars/2009-04-21-SECIE-SW-Sustainment-Lucero-brief.pdf>.
27. Carter, Ash. "Memorandum for Acquisition Professionals," Office of the Undersecretary of Defense: Acquisition, Technology, and Logistics. 14 September 2010, p. 1.
28. For more on how should-cost analysis can improve affordability, see "Should-cost Review: A Pragmatic Approach to Affordability," at <http://www.atkearney.com/index.php/Publications/should-cost-review-a-pragmatic-approach-to-affordability.html>.
29. United States Code, Title 10, Chapter 146, Section 2464, "Armed Forces: Core Logistics Capabilities," p. 1447, <http://uscode.house.gov/pdf/2010/2010usc10.pdf>.
30. Schroeder, Brian. "76th SMXG – More than just code writers," Office of Public Affairs: Tinker Air Force Base. 15 August, 2011. <http://www.tinker.af.mil/news/story.asp?id=123268153>.
31. 309th Software Maintenance Group. "Software and Hardware Engineering Solutions!" Ogden Air Logistics Center: Hill Air Force Base. Accessed 30 May, 2012. <http://www.309smxg.hill.af.mil/brochure/309SMXGbrochure.pdf>.

# Integrated Circuit Security Threats and Hardware Assurance Countermeasures

**Karen Mercedes Goertzel, Booz Allen Hamilton**

**Abstract.**Too often, software and system developers take the quality of computer hardware for granted, never doubting that the logic of the integrated circuits (ICs) on which software runs and critical application data is stored will consistently function in a dependable (correct, predictable) and trustworthy (non-malicious, non-exploitable) manner. After all, ICs seem to be free of the kinds of design and implementation flaws so common in software, and impervious to subversion by malicious code. So ICs are believed capable of achieving high levels of assurance impossible in software. This belief underpins Trusted Processor Modules (TPMs) and Hardware Security Modules (HSMs) [1], devices conceived as high-assurance platforms for critical software processes and highly sensitive data that need strong protection against tampering, interference by untrusted processes, and leakage. But is such faith in IC quality really merited? In recent years, the hardware supply chain has been flooded with counterfeit ICs of substandard quality and, more recently, hardware Trojans have emerged as a threat to the trustworthiness of IC logic. As a result, engineers of critical software-intensive systems need to employ tools that give them deeper insight into the inner workings of the ICs on which their systems' software will run. And the developers of that software need to design and implement their code so it can survive not only threats from human attackers and malicious software code, but from substandard hardware counterfeits and malicious IC logic.

## Background: Definitions

A few definitions are needed before the rest of this article will make sense:

• **Integrated Circuit (IC):** Also referred to as a semiconductor or chip. A microscopic array of electronic circuits and other components, such as components such as resistors, capacitors, diodes, and transistors, that have been diffused or implanted onto the surface of a wafer carved from an ingot of semiconducting material such as silicon. The integrated refers to the fact that all of an IC's components, circuits, and substrate material are manufactured from a single piece of silicon (by contrast with a discrete circuit in which components are manufactured separately from different materials, and later assembled). ICs range in complexity from simple logic modules and amplifiers to complex microcomputers that contain millions of circuits and components.

• **Application-Specific Integrated Circuit (ASIC):** An IC designed for a specific application, such as a specific product line of cell phones, automotive controllers, or cryptographic devices, and unable to function in any other application. ASICs may be entirely customized "from the ground up" for a specific application, customized to perform different functions within a broader general application area, or designed and produced according to a narrowly-specified design for a set of pre-manufactured devices, systems, or logic in a given platform. A few examples of ASIC applications include: cryptography using proprietary encryption and decryption algorithm, medical monitoring devices, and proprietary systems-on-a-chip (SOCs).

• **Field-Programmable Gate Arrays (FPGAs):** ICs in which the system logic can be changed after the IC has been manufactured and deployed. Because of their field-reprogrammability, FPGAs are increasingly preferred as an alternative to non-reprogrammable ICs and ASICs.

• **Intellectual Property (IP):** In the context of ICs, IP refers to the IC's design logic. This design, or IP, is applied to the IC's silicon wafer in the form of a mask (see discussion of IC manufacturing below). In the IC industry, the design is also referred to as "the silicon" for the IC (referring to the mask applied to the silicon wafer).

## Background: How ICs are Manufactured

A quick précis of how ICs are manufactured will help put the remainder of this article into context [2]. The process of manufacturing ICs typically consists of more than 100 steps—in some upward of 400. Manufacturing takes place in a clean room in which even the light sources are filtered. A single, thin slice of silicon called a wafer is created (by slicing a silicon ingot produced through a series of high-temperature physical, mechanical, and chemical processes).

The complex, interconnected design of the IC is prepared in a process similar to that used to make printed circuit boards, but with much smaller dimensions and many layers superimposed on top of each other. Each layer's design is created on a computer-aided drafting machine. The resulting design image is then projected into a "mask", which includes images for all of the dozens or hundreds of ICs to be formed on the silicon wafer. The mask is then optically reduced and transferred to the surface of the silicon wafer through a process of chemical coating and irradiation. The silicon version of the mask is transparent in some areas and opaque in others, so that when the other photolithographic and photoresist steps are applied to the wafer's surface, they collectively prepare the wafer for etching (by a chemical solution or plasma gas discharge) and/or by doping (atomic diffusion). These steps are repeated until all of the mask images created for the IC design have been permanently embedded onto the surfaces of the successive layers of the IC. Dielectric films are also applied as insulators between the layers, and on the top layer, to protect the silicon.

Once all the layers have been processed, the individual ICs on the wafer are tested for electrical functioning; those that fail are marked in ink for discard after the wafer is cut apart by a diamond saw into individual IC chips.

Each chip is then bonded onto a mounting package, which provides the IC with its contact leads—ultrathin wire leads (exponentially thinner than human hairs) connected to the package either by ultrasonic bonding or thermocompression. The mounting package is then encapsulated with a plastic coating for protection (or the chip may be assembled in a ceramic package, e.g., for certain military or aerospace applications), marked with identifying part numbers and other data, and tested again before being sealed into anti-static plastic bags for storage or shipment.

## Security Threats in IC Manufacturing and Supply Chain

As with software, IC logic is subject to design weaknesses and also to implementation flaws (or defects), which in the case of ICs are introduced during physical fabrication or manufacturing. And like software weaknesses and flaws, IC weaknesses and flaws can be exploited as vulnerabilities by human attackers. Like software, which is frequently pirated, then sold as genuine (licensed), ICs can be counterfeited with the fakes sold as genuine. Moreover, it has emerged in the last few years that like software under development by rogue programmers, ICs are also susceptible during their manufacture; in the case of FPGAs, malicious logic can be inserted even after the IC's deployment [3].

And as with software, these problems are extremely difficult to detect once the hardware has been manufactured and fielded. Indeed, the expense and time required to inspect ICs for malicious circuits or counterfeiting indicators is even greater than the cost of reviewing source code and testing executables, because of the level of expertise and the cost of specialized equipment required.

Unlike the software development lifecycle, which constitutes at most a half dozen or so stages or phases, the manufacturing process of an IC typically involves approximately 100-400 steps, each of which is susceptible to subversion by malicious actors. Such subversions may take the form deliberate design deficiencies (which, as with software, are probably only preventable through use of labor- and expertise-intensive formal methods) or malicious tampering during fabrication.

Each IC may contain as many as a billion transistors. At the rate of one transistor per second, it would take 38 years for someone to inspect all of the transistors on a single IC—an inspection process that is so difficult, tedious, and error-prone that the likelihood of finding even one tainted transistor among so many is extremely unlikely. In principle, an electronic device containing multiple ICs can be undermined by a handful of rogue transistors. This explains why ICs have become an increasingly attractive target to attackers.

Unclassified documentation of "built in" malicious logic—so-called "hardware Trojans" and "kill switches" in ICs—has yet to emerge outside of research papers. In 2007, researchers at University of Illinois at Urbana-Champaign made history by proving the feasibility of maliciously modifying hardware logic. The researchers developed two general-purpose methods for designing malicious processors, and used the resulting hardware Trojans to implement attacks that stole passwords, escalated privileges, and enabled automatic logins to computers containing the ICs. The nature of the modifications required involved the addition of a relative handful of logic gates to a pristine baseline IC. For example, the login attack that granted the researcher complete access to the targeted computer required the addition of only 1,341 logic gates to that computer's IC—0.08% of the total 1,787,958 logic gates used in the IC. In larger processors containing billions of gates, such a relatively tiny number of additions would be practically impossible to detect.

All ICs, but FPGAs in particular because they contain a significant portion of their own system-level, are vulnerable during manufacture to subversion by malicious design tools, which could be used to load a subverted design into the FPGA, in order to sabotage it (e.g., by causing it to short circuit). Unfortunately, as most hardware design-tool developers have few or no checks in place to ensure that their tools contain no such attacks on the specific functionality of ICs, the only available countermeasure at this point is to acquire only FPGAs with known-trusted cores (i.e., cores developed by trusted tools). Some FPGA manufacturers (e.g., Xilinx) digitally sign their FPGA cores to authenticate their trusted design. However, the typical FPGA chip may include multiple IP cores, both trusted and untrusted, and a digital signature used for core authentication does nothing to prevent the core's susceptibility to tampering or to snooping by other cores in the system. Interference between cores can be prevented by using a separate chip for each core; however, this approach increases power consumption and physical size, and does not prevent snooping via inter-chip communication lines on the board [4].

Automated IC test equipment can test millions of transistors per second for certain types of manufacturing fidelity. But such equipment is designed only to detect the IC's deviations from a narrow set of specifications. Any anomalies that involve aspects of the IC that are not covered by tests to verify and validate the IC against its specifications will go undetected. This not only leaves design weaknesses (especially in older IC designs), embedded hardware Trojans, "kill switches", and other misbehaviors and alterations undetected, but renders them virtually impossible to detect due to their sheer theoretical numbers. Hardware attackers often exploit the sheer complexity of modern ICs to insert their Trojan circuits, and use special or unlikely events at run time to trigger the deeply-buried malicious logic.

Inspections of suspected counterfeit ICs are somewhat more realistic. They begin with an analysis of the packaging and paperwork, then move on to several levels of inspection of the IC itself, including checking surface markings for permanency, and checking physical dimensions against known-genuine samples. Other techniques include external and internal visual analysis and radiographic inspection, material analysis, electrical testing, and accelerated life testing. Many of these tests involve specialized, often expensive equipment, such as scan electron microscopes, energy dispersive x-ray spectroscopes, Fourier transform infrared spectroscopes, s-ray fluorescence energy dispersion mechanisms, acoustic microscopes, and electrical test equipment (e.g., for Group A electrical testing and electrostatic discharge surface inspection). De-capsulation exposes the die to visual inspection under a metallurgical microscope, to reveal die markings for information such as the design year, which can then be checked with the OEM to verify whether the IC is authentic.

While IC manufacturers are likely to have some or all of the equipment necessary for IC counterfeit inspection, as such equipment is also used in IC quality and stress testing, purchasers of ICs are seldom so provisioned, nor skilled enough to use such equipment to perform the various tests. For this reason, as with the software industry, in which companies such as Veracode, Cigital, and Aspect Security (to name a very few) can be contracted to perform software analyses, an increasing number of firms (e.g., Process Sciences Inc., EQuality Process, IC Detect

Analytical Services, Silicon Cert Laboratories, ACI Technologies, Trace Laboratories) offer contract hardware analysis services to organizations that lack the resources or expertise to perform counterfeit electronics tests and analyses in-house.

The cost of fabricating ICs has driven many original equipment manufacturers (OEMs) such as Intel, Motorola, Texas Instruments, and others, to "go fabless", i.e., to outsource the fabrication and testing of their ICs to offshore foundries in countries such as China, Taiwan, South Korea, Malaysia, and the Philippines, in which labor costs are much lower. As with outsourcing of software development, this raises the question of how the fabless OEMs can assure that the ICs they received from the foreign foundry conforms exactly to the design (known as "the silicon") that they provided to the foundry—with nothing added or omitted? Moreover, increasingly OEMs are even outsourcing the design of their ICs, which raises questions about the trustworthiness not only of manufacturing, assembly, and packaging processes and tools, but of design kits and design libraries.

Because most of ICs used throughout the worldwide information and communications infrastructure are produced in unsecured facilities outside the U.S., national and homeland security establishments are increasingly concerned about the possibility of sabotage and subversion during the IC manufacturing process. However, there are also those who question how much the U.S. really has to fear with regard to subversion/sabotage of ICs or other electronic components by foreign manufacturers.

According to Martin Libicki of the RAND Corporation, "Unlike computer hacking, many of whose techniques are published on the Web and in print, the insider and component methods are essentially the province of state intelligence agencies and therefore highly protected. It is unclear how well they have worked. Consider what damage a deliberately corrupted component would have on China's reputation, much less the reputation of the guilty supplier. One discovery may create the incentive to recycle everything acquired from the now-suspect source. [It is a form] of deception and of the sort that the once-deceived is unlikely to fall for as easily again [5]."

## Security Threats Specific to Integrated Circuits

The predominant threats to the security properties of ICs are:

- Counterfeiting (threat to authenticity and often, due to deficient quality of counterfeits, dependability)
- Reverse engineering to extract IP or discover sensitive data, such as cryptographic keys, contained in on-chip memory (threat to intellectual property and data confidentiality)
- Tampering to sabotage IC operation or insert malicious functionality, such as Trojans or kill switches (threat to integrity and trustworthiness)

Each of these threats is explored below.

## IC Counterfeiting

It has been estimated that upwards of 5% of all commercial ICs are counterfeit. Counterfeiting typically refers to the production of near-identical replicas of genuine products or of product data (e.g., certificates of authenticity)—replicas that are close enough in appearance to the original to be mistaken as genuine by a typical user, reseller, tester, or other non-expert observer. The ability to copy an IC's IP (which is, remember, its design) is exploited by unscrupulous IC fabricators, particularly those offshore, for use in counterfeiting or "overbuilding". Overbuilding, also referred to as "run-in fraud", is a form of IP piracy and IC counterfeiting in which a subcontractor to an IC manufacturer copies the IP from the ICs they are subcontracted to manufacture, then inserts that IP into cheaper ICs purchased on the open market. The manufacturer then sells the ICs containing the pirated IP in direct competition with the original equipment manufacturer.

When it comes to ICs, however, most counterfeits are not replicas, but are legitimate ICs that have been altered and/or misrepresented in some way. Often, they are salvaged from discarded computer boards or electronic devices, or from the "sweepings" of IC foundries ("sweepings" are ICs rejected during manufacture, usually because they fail testing), resurfaced, and relabeled with another, newer revision number, and then delivered with documentation that misrepresents their true performance (e.g., speed, tolerance), mechanical characteristics (e.g., compliance with the Restriction of Hazardous Substances [RoHS] Directive), or pedigree (e.g., Texas Instruments vs. Fairchild Semiconductor), or ability to withstand extreme conditions, i.e., high temperatures or high clock speeds, expected of the genuine ICs that they are imitating. As a result, counterfeit chips are often more susceptible to failure or compromise than genuine ICs.

The threat of counterfeiting and overbuilding is so great that most IC OEMs have invested heavily in developing mechanisms to protect their in-chip IP. Such mechanisms include encryption, obfuscation, watermarking, and fingerprinting. Most OEMs also include mechanisms, such as bitstream encryption and authentication, for securely uploading the programming bitstreams used to reprogram FPGAs. Bitstream authentication ensures that an FPGA will accept only those programming bitstreams whose integrity can be validated via Message Authentication Codes. Some OEMs also provide their customers with authenticated remote hardware update channels that prevent the uploading of subverted update bitstreams that contain malicious design logic. In hardware obfuscation, the description or structure of the hardware is modified in a way that intentionally conceals its functionality if an attempt is made to reverse engineer the IC. Hardware IP watermarking consist of the IP owner's identifying information being embedded and concealed in the description of the IC, where it can be later detected to verify the IC's pedigree [6].

One authentication mechanism for ICs that has emerged from the research community is the Physical Unclonable Function (PUF). PUFs are unique physical characteristics that manifest as process variations in each of the ICs in a run fabricated from the same silicon mask. An IC's PUF serves as its unclonable identifier, which can be authenticated via a one-way challenge-and-response function in which the IC must correctly locate the output from one or more challenge inputs; this output should be unique to the IC, due to the uniqueness of its PUF's process variation, and thus provides the basis for unique authentication of that IC's PUF-based identifier [7].

In the end, however, there may be a far more obvious clue that an IC is counterfeit: its price. Most counterfeits, be they purses or pills or processors, sell for markedly less than the genuine article.

## Reverse Engineering

Physical reverse-engineering attacks are used to glean information about the IC's operation, and can be invasive or non-invasive. Invasive attacks, or destructive physical inspection attacks, are performed by "depackaging", i.e., partially or completely removing the packaging of the IC, either through use of acids, solvents, or other chemicals, through physical abrasion via planing, grinding, or chemical or mechanical polishing, or by evaporating the packaging material with a laser cutter. Once the IC has been depackaged, the circuitry can be scanned as each IC layer is progressively revealed through grinding. Ability to access the circuitry also enables "reconnaissance" attacks, such as reverse-engineering the circuits of the IC, or locating positions of interest to be targeted in electromagnetic attacks. In addition, the metal tracks of the IC can be probed to measure signals and voltages or to actively inject signals. A focused ion beam (FIB) can also be used to drill fine holes in the IC's insulating layer to expose the fine metal tracks without disturbing the IC's other components; a FIB can also be used to alter the IC's circuitry or to reenable disabled self-test circuitry.

Non-invasive attacks are carried out by monitoring physical properties—or signals—associated with physical phenomena that arise while the IC is running. These physical signals can be analyzed to gain information about the IC's state and the data it processes. Signals can be derived from device timing/clock rate, electrical voltage levels/power consumption (simple and differential), temperature levels, electromagnetic (EM) radiation, acoustics, and light emission. What an attacker looks for is anomalies, such as variations in power consumption or glitches in clock frequency; the attacker may also exploit the IC's detectable signals to deliberately cause errors in the device's operation. Non-invasive attacks are referred to as side-channel attacks, and cryptologists have long studied the timing, supply voltage, and electromagnetic side channels of cryptographic devices to determine whether they can be exploited to discover cryptographic keys and to detect surreptitious data leaks. More recently, researchers have investigated the use of side-channel analyses, including as gate, temperature, timing channel, and performance analyses, to detect the presence of hardware Trojans and kill switches.

While difficult to prevent, physical attacks are so technologically sophisticated, and require such substantial resources, expertise, and patience, that they remain rare. For example, while it is conceivable that focused ion beams can be used to alter the wiring and bypass security features of an IC, accomplishing this type of attack requires expensive equipment and significant knowledge, especially when targeting a modern IC that has been fabricated with nanoscale feature sizes. The exception to the "difficulty" rule involves FPGAs, which are particularly susceptible to having their IP copied in the same way that software code can be copied. Conventional SRAM-based FPGAs are particularly susceptible because their memory is volatile, which means it must be re-initialized every time power is applied. Each re-initialization requires an external bitstream to be loaded into the FPGA. That external bitstream provides an easily exploitable, non-invasive conduit by which the FPGA's IP can be captured and copied.

The most secure FPGA has a single chip, with the non-volatile memory located on the FPGA chip itself. The FPGA's strong encryption capability is used not only for encrypting IP and programming bitstreams, but also the data in the on-chip memory. The non-volatile memory registers also store the encryption keys and the identifiers used to authenticate bitstreams. Encryption also protects IP and data stored in FPGAs that are subjected to physical "sand-and-scan" reverse engineering or data extraction attacks [8].

## Tampering

Tampering to alter the functionality of an IC other than an FPGA is always done to the design of the IC, because it is virtually impossible to tamper with fabricated chips in a way that is fine-grained enough to alter the hardware's logic without simply destroying the hardware. Post-manufacture tampering is a greater concern for FPGAs whose system programming can only be modified safely if certain secure IC programming and data protections are provided to control access to the FPGA's IP and the data stored in its on-chip memory [9].

Due mainly to IC manufacturers' concerns over physical tampering to extract IP, an increasing number of ICs now have countermeasures against physical attacks built in. For example, the IBM 4758 co-processor "wraps" its hardware within a tamper-sensing and tamper-reactive "shell". Others have their IP encrypted so that even if the IC is physically attacked, its IP cannot be deciphered by the attacker [10]. Techniques for obfuscation of logic in ICs have also emerged, and are being improved upon to strengthen IC self-protection against intellectual property reverse-engineering [11].

Several interesting anti-tamper mechanisms have been emerging from DoD's Anti Tamper (AT) research initiative (the focus of which is to develop technologies that can prevent reverse engineering and extraction of IP from ICs used in sensitive DoD systems and applications) [12]. One such mechanism is IC metering, which provides a set of security protocols designed to enable an IC design house to maintain control of an IC after its fabrication. Such control may be passive, such as and may constitute limiting the number of ICs fabricated and the properties they exhibit, or it may be active, such as building into the IC the capability to automatically disable itself at run-time if any indication of tampering is detected [13].

Like time bombs and logic bombs in software, the intentional corruption of hardware generally occurs during its design, implementation, or manufacturing—well before the malicious logic is activated. But unlike software, with the exception of FPGAs, sabotaged ICs cannot be patched, so they remain a threat indefinitely. Remediating well-crafted IC-level vulnerabilities or malicious insertions would likely require physically replacing the compromised hardware. The skill required to replace hardware, particularly in deeply embedded systems, would ensure that compromised ICs remain in active use even after the discovery of the vulnerability or Trojan.

Also, because the IC represents the lowest layer in the computer system, malicious logic at the IC level can provide a means to bypass, subvert, or gain control over all software running above it—allowing sophisticated and stealthy attacks to be crafted specifically to evade software-based defenses. Any "defense in depth" that involves only multiple layers of protection implemented by software, even low-level software such as VMs and kernel-level processes, can at best deter but not prevent attacks that originate from within the processor on which the software is installed.

For example, attackers might introduce a sequence of predetermined bytes into the IC to activate embedded malicious circuits, enabling leaks of highly-sensitive data or cryptokeys, to halt the processor at critical or random processing times, to scan for electromagnetic signals that provide the external cues for processor shut-downs, or to facilitate reverse engineering of the IC design. More sophisticated hardware Trojan logic has been devised that enables attackers to escalate privileges, turn off access control checks, and execute arbitrary instructions, thereby gaining a path to taking control of the machine, and establishing a foothold for subsequent system-level attacks. An IC with such a hidden Trojan circuit installed in a firewall could facilitate remote exploits; e.g., a packet sent from a predetermined network address or a key encoded as a series of requests to different ports could be used as the trigger for the Trojan to "reset" the firewall, thereby granting full unprotected access to the network [14].

But because the results of many hardware attacks manifest identically to "normal" hardware failures, such attacks may be misattributed to manufacturing defects or design flaws rather than malicious logic.

## Hardware Assurance for ICs

As a recognized discipline, hardware security assurance—as distinct from hardware quality assurance or system safety assurance—is at a point in its maturity comparable to that of software security assurance a decade ago. Aside from antitamper mechanisms for device-level hardware such as smart cards, cryptographic devices, trusted processor modules, and hardware security modules (used in automatic teller machines), hardware security assurance largely focuses on ICs and IC assemblies, though until very recently, the main focus has been anti-counterfeiting and IP protection, including the application of post-silicon validation techniques [15] to post-manufacturing discovery of flaws in ICs indicative of counterfeiting or malicious circuits.

With the recent proof by researchers of the practicability of hardware Trojans, however, the need for a hardware manufacturing counterpart to a secure software development lifecycle has become clear. If this growing awareness and concern over malicious hardware follows a similar trajectory to that for malicious software, we should expect the discipline of hardware security assurance to further coalesce and advance in the next few years.

Because most IC manufacturing is done outside of the United States and Europe, predominantly in China, U.S. and NATO buyers of ICs lack both visibility into and control over the supply chain for the ICs on which they so heavily rely. Depending on the criticality of the purpose to which ICs will be put, reliance on commodity ICs may simply be too risky, even if unambiguous knowledge of IC pedigree and provenance throughout the supply chain could be obtained (which it cannot). When total control over and visibility into production of ICs for critical applications are vital, alternatives to commodity ICs can be designed and fabricated by trusted foundries.

In the U.S. by the Department of Defense (DoD), the Department of Energy (DOE), and the National Aeronautics and Space Agency (NASA) have already made significant progress in hardware assurance for ICs through their trusted foundry initiatives: DoD's Trusted Foundry Program, administered by the Trusted Access Program Office at the National Security Agency (NSA), with foundry accreditations performed by the Defense Microelectronics Activity (DMEA) [16], and DOE's Trusted Foundry at the Sandia National Laboratory Microsystems Center. In addition, research programs such as DoD AT and the Defense Advanced Research Projects Agency's (DARPA's) Trust in Integrated Circuits program [17] focus on advancing the technologies needed to increase trustworthiness of ICs intended for mission critical, security critical, and safety critical applications (e.g., cryptographic devices, weapon systems, nuclear power plants).

DoD AT and DARPA Trust in ICs are producing tamperproof and tamper-resistant IC technology that, while it may not scale to extremely high-volume manufacturing general-purpose ICs from companies such as Intel and Motorola, has successfully transitioned into commercially-produced ICs with anti-copying, antitamper, anti-reverse engineering IP protections, such as CPU Tech's Acalis [18] and Altera's Stratix and Arria [19] FPGAs.

Outside the U.S., significant advances in hardware security assurance are being made in the private sector—specifically in the payment and bank card industry, with particular focus on the security of ICs used in smart cards.

## Conclusion

At a conceptual level, security assurance deficiencies in the manufacturing and supply chain for logic-bearing integrated circuits are directly comparable to security assurance deficiencies in the software development life cycle and supply chain, and like software security assurance deficiencies, hardware security assurance deficiencies in IC production have troubling implications that go far beyond concern over reduced quality. Intentional threats to ICs, both in- and post-production, threaten the dependable, trustworthy operation not only of the ICs themselves, but of any embedded and non-embedded software-intensive systems in which they are a core component. Threats to IC security manifest as counterfeiting (threat to IC authenticity, and by extension dependability), tampering and malicious circuitry (threats to IC trustworthiness), and reverse engineering (threat to confidentiality of intellectual property and sensitive data).

The dependability and trustworthiness of all hardware functions on which critical software relies should be verified and validated. Pre- and post-silicon IC testing tools and techniques have emerged—and continue to emerge—for detecting indicators of counterfeiting indicators and malicious inclusions in ICs. Unlike software testing, however, IC tests are unlikely to be within the abilities of system testers; instead, expert test labs will have to be enlisted. Alternately, assurance of ICs can be achieved through acquisition from verifiably trustworthy IC suppliers, be they accredited trusted foundries or commodity suppliers with a demonstrated commitment to secure IC production, which includes the full range of necessary hardware assurance processes and tests, as well as full supply chain traceability and transparency.

Ideally, all ICs destined for use in critical systems would be individually tested, and if they failed any hardware security assurance test, rejected. In reality, 100% test coverage for all ICs may be possible only in systems in which ICs will be deployed in (very) limited quantities. For most system deployments, only a sampling of ICs (one hopes, large enough to be meaningfully representative) can possibly be tested. In such cases, trends analysis of results across the ICs in the tested sample will help the tester determine whether a tendency towards failure is likely to appear across the entire lot of ICs, and whether it may indicate a broader systematic deficiency in the manufacturing or supply chain practices of a particular IC vendor.

Testing as early in the system development lifecycle as possible will allow time and maximum scope for IC replacement (at the individual IC or whole-lot level) or supplier substitution. Moreover, critical systems must be architected to enable dynamic replacement of failed/suspect hardware, ideally with minimal operational disruption.

Furthermore, software developers should become knowledgeable about the threats to IC security, and the deficiencies in hardware security assurance practices in IC manufacturing that make ICs susceptible to those threats, so that they can design and implement critical software to include countermeasures that can mitigate the potential impacts of defective and anomalous hardware operations, so their software can survive any failures or subversive hardware operations that may originate from counterfeit or malicious ICs.

## Further Reading

Goertzel, Karen Mercedes, *et al. Security Risk Management for the Off-the-Shelf (OTS) Information and Communications Technology (ICT) Supply Chain.* Herndon, VA: Information Assurance Technology Analysis Center, 17 August 2010.

Tehranipoor, Mohammad. “Mohammad Tehranipoor on Integrated Circuit Security”. *YouTube*, 3 May 2012. <http://www.youtube.com/watch?v=d9Ib4s7sHWM>

University of Connecticut Center for Hardware Assurance, Security, and Engineering (CHASE) Website. <http://www.chase.uconn.edu>

EuroSmart. *Security IC Platform Protection Profile* (Version 1.0, BSI-PP-0035, 15 June 2007). <http://www.commoncriteriaportal.org/files/ppfiles/pp0%20035b.pdf>

European Joint Interpretation Working Group (JIWG). *The Application of CC to Integrated Circuits*, Version 3.0 Revision 1, CCDB-2009-03-002, March 2009. <http://www.commoncriteriaportal.org/files/supdocs/CCDB-2009-03-002.pdf>

## ABOUT THE AUTHOR

Karen Mercedes Goertzel, CISSP, is an expert in application security and software and hardware assurance, the insider threat to information systems, assured information sharing, emerging cybersecurity technologies, and information and communication technology (ICT) supply chain risk management. She has performed in-depth research and analysis and policy and guidance development for customers in the U.S. financial and ICT industries, DoD, the Intelligence Community, Department of State, NIST, IRS, and other civilian government agencies in the U.S., the UK, NATO, Australia, and Canada.

**Lead Associate**
**Booz Allen Hamilton**
**7710 Random Run Lane - Suite 103**
**Falls Church, VA 22042-7769**
**Phone: 703-698-7454**
**E-mail: goertzel_karen@bah.com**

## REFERENCES

1. Fisher, David A., et al. “Trust and Trusted Computing Platforms”. Carnegie Mellon University/Software Engineering Institute Technical Note CMU/SEI-2011-TN-005, January 2011. http://www.cert.org/archive/pdf/11tn005.pdf -and- Edmison, Joshua N. *Hardware Architectures for Software Security.* Virginia Polytechnic Institute and State University Ph.D. Dissertation, 29 June 2006. <http://scholar.lib.vt.edu/theses/available/etd-10112006-204811/unrestricted/edmison_joshua_dissertation.pdf>
2. “Integrated Circuit”. *Made How, Volume 2: How Products Are Made.* <http://www.madehow.com/Volume-2/Integrated-Circuit.html> -and- Intersil, “How Semiconductors are Made”. <http://rel.intersil.com/docs/lexicon/manufacture.html>
3. Clark, Wesley K., and Peter L. Levin, “Securing the Information Highway”. *Foreign Affairs* (November/December 2009). <http://www.international.ucla.edu/burkle/news/print.asp?parentid=112702>
4. King, Samuel T., et al. “Designing and Implementing Malicious Hardware”. *Proceedings of the First USENIX Workshop on Large-Scale Exploits and Emergent Threats* (LEET). San Francisco, CA, 15 April 2008. <http://static.usenix.org/events/leet08/tech/full_papers/king/king.pdf>
5. Libicki, Martin C. *Cyberdeterrence and Cyberwar.* Santa Monica, CA/Arlington, VA/Pittsburgh, PA: The Rand Corporation, 2009. <http://www.rand.org/pubs/monographs/2009/RAND_MG877.pdf>
6. Huffmire, Ted, et al. “Managing Security in FPGA-Based Embedded Systems. *IEEE Design and Test of Computers* (Volume 25 Issue 8, November/December 2008).
7. Helinski, Ryan. “Physical Unclonable Functions”. 27 October 2009. <http://www.ece.unm.edu/~jimp/HOST/slides/RyansPUFslides.pdf>
8. Mertz, Michael. “Secure in-system programming for FPGAs”. *EE Times* (26 October 2005). <http://www.eetimes.com/design/programmable-logic/4014793/Secure-in-system-programming-for-FPGAs>
9. Landis, Dave. “Programmable Logic and Application Specific Integrated Circuits” (course notes). <http://cset.sp.utoledo.edu/cset4650oc/fpga_arch_intro.pdf>
10. Kastner, Ryan, and Ted Huffmire. “Threats and Challenges in Reconfigurable Hardware Security”. *Proceedings of the 2008 International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA 2008).* Las Vegas, NV, 14-17 July 2008. <http://cisr.nps.edu/downloads/08paper_threatschallenges.pdf>
11. Rajendran, Jeyavijayan, et al. “Security analysis of logic obfuscation”. *Proceedings of the 49th ACM Annual Design Automation Conference (DAC ’12).* San Francisco, CA, 3-7 June 2012.
12. DoD Anti-Tamper Program Website. <http://at.dod.mil>
13. Koushanfar, Farinaz. “Integrated Circuits Metering for Piracy Protection and Digital Rights Management: An Overview”. *Proceedings of the 21st Association of Computing Machinery (ACM) Great Lakes Symposium on Very Large-Scale Integration (GLSVLSI ’11).* Lausanne, Switzerland, 2-4 May 2011. <http://aceslab.org/sites/default/files/GLS-VLSI.pdf>
14. Abramovici, Miron, and Paul Bradley. Integrated Circuit Security–New Threats and Solutions. *Proceedings of the Fifth Annual Cyber Security and Information Intelligence Research Workshop (CSIIRW ’09).* Oak Ridge, TN, 13-15 April 2009.
15. Mitra, Subhasish, et al. “Post-Silicon Validation Opportunities, Challenges and Recent Advances”. <http://www.eecs.berkeley.edu/~sseshia/pubdir/postSi-dac10.pdf>
16. DoD Trusted Foundry Program Website. http://www.trustedfoundryprogram.org/index.php -and- U.S. Department of Commerce. Chapter VIII: Fabrication and Design of National Security Products. *Defense Industrial Base Assessment: U.S. Integrated Circuit Fabrication and Design Capability,* May 2009, pages 92-99. <http://www.bis.doc.gov/defenseindustrialbaseprograms/osies/defmarketresearchrpts/bis_ote_ic_report_051209.pdf>
17. Adee, Sally. “The Hunt for the Kill Switch”. *IEEE Spectrum* (May 2008). <http://spectrum.ieee.org/semiconductors/design/the-hunt-for-the-kill-switch> -and- Robinson, Brian, “Building Trust into Integrated Circuits”. Defense Systems (February 2008). <http://defensesystems.com/articles/2008/02/building-trust-into-integrated-circuits.aspx>
18. Adee, Sally. “New Chip Brings Military Security to Commercial Processors”. *IEEE Spectrum* (April 2009). <http://spectrum.ieee.org/computing/hardware/new-chip-brings-military-security-to-commercial-processors>
19. Altera. “FPGAs: About Stratix Series–Design Security”. <http://www.altera.com/devices/fpga/stratix-fpgas/about/security/stx-design-security.html>

CROSSTALK
The Journal of Defense Software Engineering
Vol. 24 No. 2
Jan / Feb 2013
Vol. 26 No. 1
SOFTWARE PROJECT MANAGEMENT
LESSONS LEARNED
July / August 2011
Vol. 24 No. 4
Jan / Feb 2012
Vol. 25 No. 5
March / April 2013
Vol. 24 No. 1
Nov / Dec 2010
Vol. 23 No. 6
Enjoy!

# Upcoming Events

Visit <http://www.crosstalkonline.org/events> for an up-to-date list of events.

**Technology Tools for Today (T3) Conference**
3-5 November 2013
Rosemont, IL
http://2013t3enterprise-eorg.eventbrite.com/

**Aircraft Survivability Symposium 2013**
5-7 November 2013
Monterey, CA
http://www.ndia.org/meetings/4490/Pages/default.aspx

**2013 Homeland Security Symposium**
7-8 November 2013
Washington, DC
http://www.ndia.org/meetings/4490/Pages/default.aspx

**OWASP AppSec USA 2013**
18-21 November 2013
New York, NY
http://www.sourcesecurity.com/events/free-event-listing/owasp-appsec-usa-2013.html

**29th Annual Computer Security Applications Conference**
9-13 December 2013
New Orleans, LA
http://www.acsac.org/2013/cfp/cfp.html

**Winter 2013 Software & Supply Chain Assurance (SSCA) Working Group Sessions**
17-19 December 2013
McLean, VA
https://buildsecurityin.us-cert.gov/swa

**International Conference on Computing, Networking and Communication**
2-6 February 2014
Honolulu, HI
http://www.wikicfp.com/cfp/servlet/event.showcfp?eventid=30749&copyownerid=548

**2014 T3 Advisor Conference**
10-12 February 2014
Anaheim, CA
http://2014t3-eorg.eventbrite.com/

**Spring 2014 Software & Supply Chain Assurance Forum**
11-13 March 2014
McLean, VA
https://buildsecurityin.us-cert.gov/swa

**30th Annual National Test & Evaluation Conference**
24-27 March 2014
Seattle, WA
http://www.ndia.org/meetings/4910/Pages/default.aspx

**Summer 2014 Software & Supply Chain Assurance (SSCA) Working Group Sessions**
24-26 June 2014
McLean, VA
https://buildsecurityin.us-cert.gov/swa

# Better Never Than Late?

**This summer** was full of travel for me. In fact, I put more than 10,000 miles on my car. Since May, I have gone through three different GPS units. To begin with, I have one built into my car. It is convenient to use (push a button and it responds to voice commands, and it even mostly understands my Texas drawl). However, it does not display some data that my older top-of-dash GPS displayed. Mainly, it does not display the current speed limit, nor will it flash red when I am speeding—a necessary feature for me. It is also not at eye level. Finally, when I am using the XM radio, the GPS screen is not shown (although it still gives me voice commands if I want). I still like my top-of-dash portable GPS. My older portable

GPS quit working, so I bought a new unit which is also voice activated (thank goodness, using different "trigger words" to activate than the built-in GPS). When not using the XM radio, I use the built-in unit to show an "overview" map on a large scale, with distance and time to destination. I use the dash-top unit to show small-scale streets and turns, along with speed limit and my current speed. I know it sounds confusing, but it works for me. I can use it without thinking—the mark of a good system. (And yes, I know I am a geek. Please, no emails!)

You would now expect me to expound about how the two GPS units disagree as to route, causing confusion. In fact, about 99.9% of the time, they totally agree on the route—and when they disagree, it is usually over trivial details. What irritates me, however, is the lack of consistent accuracy of the maps. Several times, both units will cheerfully tell me that the restaurant or hotel I am looking for is, "100 yards on the right" when it is actually on the left. 100 yards (at 45 mph) is not enough time to try and change several lanes in traffic.

Obviously, both GPS units (different manufacturers) have the same map data. Both are usually wrong at the same time. Having the wrong data sort of leads me to what I consider the first law of information assurance:

1st Law—The information should be accurate.

Getting the information too late to do anything about it leads to the second law, merging information assurance and delivery time:

2nd Law—The information needs to also reach me in time to be useful.

Getting the destination information 50 yards before I need it—possibly not useful. Getting it 400 yards early? That is probably plenty of time, unless the data is wrong. When the data is wrong, I need the information early enough so that I can realize that it is incorrect, so that I can make corrective action.

If the information might be incorrect, I need to be notified early, so I can potentially take alternative action.

The problem is that I tend to totally rely on my GPS when driving, and I no longer really think that much about my destination—I let the GPS do the work. Sort of like I no longer need to remember phone numbers (that is what my smartphone is for).

However, if I know that the GPS is going to be wrong (construction zones and detours), well, I have a few minutes to "get my bearings", and put my brain back into gear. I can usually manage to get back on route. It would be nice if occasionally the GPS would say, "I am not really sure about the next 2.7 miles. Try looking at the detour signs yourself."

I was mentioning this problem with lack of useful information in a timely fashion with my wife, Deb. Specifically, while we were driving 1.4 miles down a "possibly unpaved road" (the GPS got the unpaved part right!) attempting to get back on Interstate 40 in West Memphis, Tennessee. I got a bit lost when the GPS gave me incorrect directions in a detour around a construction zone. My wife, a former Digital Equipment employee back in the good old days, reminded me of the term GIGO—Garbage In, Garbage Out. I had not thought of that term in a while but it still explains a lot. Bad information input = problems with the output. What does "bad information" mean? It could mean simply out-of-date information, or it could also mean malicious (and potentially destructive) data. Either way, the system fails.

It is hard getting valid information out in time. Frankly, I do not know how they encode all of the map data and access it in "close to real time." I am pretty impressed. All I have to do is say, "Hey GPS," then tell it to find the closest coffee shop, and there it is, complete with routing. Need the next rest area? Next gas station? Next ATM? Next restaurant? All just a few words away. Life is pretty good, the GPS is mostly accurate, and I maintain a full coffee cup most of the trip.

All of which means that I drove 11,327 miles this summer, at an average speed of 51 mph, and achieved 53.3 mpg (yeah—I am one of those hybrid owners—but I am NOT one of slow ones who drive 55 in the right lane—I zip along at 70). And I did this with mostly correct GPS directions delivered in mostly plenty of time to be useful.

What if the information was wrong or slow? Well, it is just a car trip. After all, there are route signs. A wrong turn, at worse, usually means less than a 5-minute detour. In the grand scheme of life, this is a pretty minor thing. I can always turn around and try again!

Of course, if the data were wrong or late, and the application involved a supersonic jet, a satellite navigation program, some type of munitions guidance system, an air traffic control program, or a missile defense program, I suspect the consequences would be much worse than a 5-minute detour driving 1.4 miles down a, "potentially unpaved road."

That is why I sometimes do not sleep well at night.

**David A. Cook, Ph.D.**
**Stephen F. Austin State University**
**cookda@sfasu.edu**

CIVILIAN TALENT IS MISSION-CRITICAL.

**LET'S GET TO WORK.**

Work for Naval Air Systems Command (NAVAIR) and you'll support our Sailors and Marines by delivering the technologies they need to complete their mission and return home safely. NAVAIR procures, develops, tests and supports Naval aircraft, weapons, and related systems. It's a brain trust comprised of scientists, engineers and business professionals working on the cutting edge of technology.

You don't have to join the military to protect our nation. Become a vital part of NAVAIR, and you'll have a career with endless opportunities. As a civilian employee you'll enjoy more freedom than you thought possible.

Discover more about NAVAIR. Go to **www.navair.navy.mil**.

Equal Opportunity Employer | U.S. Citizenship Required

NAVAIR CIVILIAN

CHOICE IS YOURS.

CrossTalk thanks the above organizations for providing their support.